geosciences

MDPI

Article

OGC-to-W3C Services: A Wrapper-Based Solution for Geospatial Metadata Exchange

Pasquale Di Giovanni [1,2], Michela Bertolotto [1,*], Monica Sebillo [2] and Giuliana Vitiello [2]

[1] School of Computer Science, University College Dublin, Dublin, Ireland; digiovannipasquale@gmail.com
[2] Dipartimento di Informatica, Università degli Studi di Salerno, 84084 Fisciano, Italy; msebillo@unisa.it (M.S.); gvitiello@unisa.it (G.V.)
* Correspondence: michela.bertolotto@ucd.ie

Received: 12 April 2018; Accepted: 19 June 2018; Published: 22 June 2018

Abstract: When trying to compose services that are developed according to different standards, interoperability issues arise that are often faced by developing ad-hoc solutions. A typical example is represented by the composition of W3C and OGC services, which were developed more or less concurrently by independent organizations. In such a context, developing a wrapper that abstracts as much as possible the inner details and logic of a wrapped service well embraces the low-coupled nature of the general Service Oriented Computing paradigm. This paper focuses on the proper management of geospatial metadata into a W3C-based environment and is based on the development of a wrapper that exposes OGC functionality in a W3C-compliant way, thus overcoming the technical differences between the two types of services. To efficiently accomplish this task, a flexible, wrapper-based, and standard-compliant mechanism for metadata retrieval is proposed that avoids the use of third-party entities. The proposal exploits the scalable and adaptable nature of the WS-Metadata (Web Services Metadata) specification.

Keywords: services composition; geospatial metadata; metadata exchange; W3C standards; Open Geospatial Consortium

1. Introduction

Nowadays, we are witnessing an explosive growth of software applications, which, by aggregating and harnessing the rich information available from highly diverse data sources, are transforming the way we use information in our daily activities. The increasing need to easily access and efficiently process heterogeneous data available almost everywhere has a deep impact on the design of software solutions that could promote a gradual transition from stand-alone and centralized architectures to distributed ones. Despite its undeniable advantages, the design of a distributed system is still a very complex activity with several issues impacting on all proposed solutions. Among such challenges, the heterogeneity of data representations and the related invocation mechanisms that may characterize each implementation, directly affect the ability of final users to successfully retrieve and exploit the desired information.

The above-mentioned issues have a considerable impact also on one of the current leading approaches for the development of distributed solutions, namely Service Oriented Computing (SOC). The core idea behind the Software as a Service (SaaS) model is the concept of Service, a self-describing computational element that, exposing its public interface using open standards, supports the development of distributed applications [1]. In particular, the Service-oriented Architecture (SOA), an architectural approach that is compliant with this model, is mainly useful when different software solutions, developed with various technologies, need to communicate with each other [2]. Indeed, there exist several complex realities where the adoption/integration of new paradigms may prove slow

and sometimes impossible. Legacy systems, such as information systems from public administrations and medium/large enterprises (e.g., hospitals and banks), are examples of such infrastructures. In those cases, also due to security issues, data exchange is controlled by and limited to a specific set of packages. In those solutions (almost), all medium-sized services are exposed along with documents and protocols addressed to specify how to interact with them (in terms of available functionality, constraints, and catalog/registry for accessing them.) Data is packaged and exchanged between systems that only need to know how to prepare and recover data to be transferred. Once received, data is internally handled, processed, and possibly sent to different receivers (also to other Web services), thus making the service-based solutions the backbone of a growing number of modern information systems.

In order to achieve the practical fulfilment of those service-based solutions, every entity that is involved in the computation must adhere to a set of globally accepted standards that regulate the fundamental aspects concerning the structure of a service. In this context, the World Wide Web Consortium (W3C) has established a series of proposals whose specification and implementation rely on the use of the Extensible Markup Language (XML) in order to guarantee their independence from a specific platform [3]. One of such proposals is the Web Services Description Language (WSDL) [4] for the description of a service public interface and another is the SOAP (Simple Object Access Protocol) protocol [5] for the communication among services.

The majority of current SOA-based solutions then rely on the W3C specifications. However, a quite significant exception is represented by the choices that are made by the geospatial community. Under the guidance of the Open Geospatial Consortium (OGC), it has defined its own set of standards that is explicitly tailored to the specific characteristics of geospatial data [6]. Such standards are based on different design assumptions and turn out to be not fully compatible with the W3C ones. One of the historical and practical reasons for the definition of two different proposals is the fact that, in the early stages, the two communities devoted their efforts to developing standards independently of each other (and more or less concurrently). In addition, each of those communities' efforts was aimed at supporting the needs of their main users by taking into account the characteristics of data and functionality required by them. As a result, the two sets of standards present several dissimilarities, which make a seamless communication not directly achievable. Thus, applications dealing with multifaceted information need to individually interact with different and potentially conflicting service-based systems. All of the retrieved data have to be manually assembled on the client side, thus adding a significant computational overload. In resource-limited contexts, such as mobile computing, which may be simply unfeasible.

Given the importance of the geospatial component within the increasing amount of available data, the ability to manipulate data by using a single approach, and, above all, the possibility to exploit the flexibility of services composition to obtain aggregated information directly on the back end still remain crucial goals.

In 2015, the W3C announced that W3C and OGC would collaborate to "Integrate Spatial Data on the Web" (https://www.w3.org/2015/01/spatial). However, the scope of this collaboration relates to the integration of data rather than services and the service integration still remains an open issue. Although more recently the W3C has published best practice guidelines for the definition of metadata associated with spatial datasets (https://www.w3.org/TR/sdw-bp/#bp-metadata), they do not apply to legacy data and services.

Since the dissimilarities between OGC and W3C proposals affect all the key sections of service development, one of the few options to let existing services communicate consists of mapping one set of standards onto the other at both the syntactic and semantic levels. In particular, although the mapping can be either from W3C to OGC standards or vice-versa, the best option seems to be the adaptation of the OGC standards to the W3C ones. Several reasons support this choice. First of all, W3C standards constitute the basis for the majority of current SOA-based industrial solutions. Therefore, there is a huge supporting infrastructure that can be exploited to manage and share

geospatial information, thus promoting its use by a wider range of users. Moreover, the adoption of the fundamental W3C standards for interface definition and messages exchange has been investigated by important international initiatives, including the Infrastructure for Spatial Information in the European Community (INSPIRE) [7]. The OGC itself has set a special working group to provide general recommendations and guidelines for adding W3C support to existing and future OGC services. Finally, with a deeper integration of the two worlds, one of the fundamental standards for the composition of W3C services, namely, the Web Services Business Process Execution Language (WS-BPEL) [8], can be used for the management of the services workflow.

The research that we are carrying out in this field aims to design data harmonization processes that integrate and manipulate heterogeneous data guaranteeing the interoperability requirement, central for systems and devices that need to exchange data and interpret the shared data, so that it can be understood by users. In the paper, we investigate a solution to the geospatial data exchange among systems that are built according to different standards and software technologies, taking into account the role that metadata play for the actual exploitation of both geospatial information in general and OGC services. In particular, we focus on the proper management of geospatial and OGC metadata in a W3C-oriented infrastructure, while addressing syntactic features to guarantee the interoperability of data exchange process.

A feasible way to realize such an OGC-to-W3C mapping is represented by the development of a wrapper, a software module whose main purpose is to query an OGC service and return the geospatial information in a W3C compliant format while keeping the structure of the original W3C or OGC services unchanged. The idea of developing a wrapper has already been discussed in the literature [9–11]. However, a suitable solution has not been implemented and validated, as its actual fulfillment represents quite a complex task that cannot be automated by a straightforward mechanical translation process. In fact, several issues need to be carefully taken into account during the wrapper design in order to make this solution a viable option. In particular, to represent a useful solution, a wrapper should guarantee, first of all, a seamless exchange of geospatial information between OGC and W3C services. Moreover, to be effectively deployable in a real-world infrastructure, an implementation of such an option should not modify the behavior and semantics of the underlying standards and should be totally transparent for those services that do not deal with geospatial information.

The remainder of the paper introduces and investigates the proposed approach for the development of an OGC-to-W3C wrapper with the ability of exposing OGC metadata exploiting only W3C-compliant standards. Section 2 recalls some background concepts. After a brief overview of the two main W3C standards, the focus is on the main characteristics and issues of an OGC-to-W3C wrapper. Then, it describes the metadata management in the context of the SOC paradigm and analyzes the various available options for their actual retrieval. The focus is on key differences between the standard ways to directly retrieve them from the intended services in W3C and OGC environments, respectively. In Section 3, the proposal is illustrated to provide a W3C standard that deals with the direct metadata retrieval with the support for OGC metadata. Section 4 discusses the technical feasibility of the proposal by showing how one of the most used software implementations of the W3C services stack can be modified to seamlessly support the new metadata types. Some conclusions are drawn in Section 5.

2. Background

A critical aspect during the development of a software solution for mapping OGC standards onto W3C ones at the syntactic level relates to the proper management of geospatial and OGC metadata in a W3C-compliant architecture. The design philosophy for metadata exploitation and exchange in those environments is very different. For OGC services, exposing their metadata represents a mandatory task and every service must provide, through its public interface, a well-standardized operation for metadata retrieval (GetCapabilities). W3C services, instead, are more flexible and can use several

options to expose their metadata. In order to offer a seamless interoperability, an approach consists of exposing OGC metadata by using the current W3C standards.

In this Section, after a brief overview of the two main W3C standards that constitute the building blocks of the whole W3C infrastructure, the main characteristics and issues of an OGC-to-W3C wrapping are summarized. In the second part, the metadata management in a SOA context is discussed, and the various available options are then analyzed for their actual retrieval.

2.1. Main Concepts and Related Work

Nowadays, the majority of service-based solutions rely on the proposals of the W3C. In this context, the cornerstones of the whole W3C services stack are WSDL and SOAP [5]. WSDL is an XML based language for describing W3C services and how to access them. A WSDL document separates the abstract aspects of a service description from the concrete aspects, such as the binding with a certain network protocol, thus preserving the public interface of the service by changes in the underlying technology. SOAP is an XML-based protocol for the exchange of information in a decentralized, distributed environment. It consists of an envelope describing the message content and the way to process it, a set of encoding rules for expressing data types instances, and a convention for representing remote procedure calls and responses [5]. Its main design goals are simplicity and extensibility, and are most widely used in conjunction the Hypertext Transfer Protocol (HTTP). Three basic components characterize a typical SOAP message: an Envelope, which can be seen as the container of the message itself, a Header, which contains important information about how the message should be processed, and a Body that contains the actual information of the SOAP message intended for the ultimate recipient of the communication. This latter can be any well-formed XML content, provided that such a content has a namespace and it does not contain any processing instruction or Document Type Definition (DTD) reference [2].

Despite the success of W3C services and the huge amount of available documentation and frameworks for services development, the geographic community has developed a broad consensus around the OGC services, the de facto standard concerning the exchange of geospatial data in distributed environments. OGC is an international industry consortium, including companies, government agencies, and universities "participating in a consensus process to develop publicly available interface standards". Although the overall theoretical principles are the same, the actual design of geospatial services is quite different when compared to the choices made by W3C for their services stack. The Consortium has specified standards and guidelines for the Data and Metadata management and Interface definition of geospatial services. The Data standards define the storage file format of geospatial data, the Metadata information describe both geospatial data and geospatial services and the Interface specifications govern the way that services and potential clients exchange information. As for the actual dissimilarities between the two sets of standards, the first important difference is represented by the strong standardization that is imposed by OGC on the public interface of a geographic service. Moreover, differently from W3C services, OGC services share some common operation request and response contents, as well as some parameters for operation request and response. As for the service specification, the most widespread and commonly used are the Web Map Service (WMS) to request georeferenced map images from a geospatial database, the Web Feature Service (WFS) for accessing and manipulating geographic features, and the Web Coverage Service (WCS) to define an interface for the exchange of geospatial information representing phenomena, known as coverages. The only functionality common to these three types of services is GetCapabilities, which allows for a geospatial service to expose its capabilities to clients.

To overcome the syntactical and semantic differences between W3C services and OGC services, recent literature shows a general consensus on the development of a service wrapper that leaves unchanged the structure of existing services. In a SOA environment, such a type of software module is usually a service itself requiring the typical supporting infrastructure of service-based

solutions. In particular, for an OGC-to-W3C wrapper, its typical workflow is constituted by four main steps [12,13]:

1. the wrapper receives from a W3C service a message containing a request for a specific geospatial dataset;
2. the wrapper translates the W3C-based request into a format suitable for the underlying OGC service, and sends the query;
3. the OGC service returns the desired information; and,
4. the wrapper translates the received response into a W3C-compliant format and sends it back to the requesting client.

By implementing such steps, possible solutions can be partitioned into three broad categories:

1. use of the SOAP protocol for the actual exchange of geographic information;
2. use of the WSDL standard to describe the public interface of a generic OGC service; and,
3. proper management of the fundamental geospatial metadata in a W3C standards oriented infrastructure.

The focus of our research relates to the third issue, i.e., proving a W3C standard that deals with the direct metadata retrieval with support for OGC metadata. In order to introduce our proposal, the following subsection is focused on a brief discussion on the metadata management in a SOA context. The analysis of the various available options for their actual retrieval is also provided and the key differences between the standard ways to directly retrieve them from the intended services within both of the environments are finally discussed.

2.2. The Primary Role of Metadata in SOA

Metadata have gained a primary role in every key-aspect of a SOA where they provide for fundamental support in the whole life-cycle management of a complex system. In particular, the core functionalities that extensively make use of metadata include the initial retrieval of all the information (such as the list of capabilities, access rules, additional policies) that allows for a generic client both to suitably interact with a service and begin message exchange. However, to properly accomplish this task, it is necessary to overcome two fundamental issues: how to provide clients with such information, and how to obtain the same interpretation by all involved entities.

As for the first issue, several possible solutions are available [2,14]. While, OGC [15] proposes guidelines to retrieve spatial data and services metadata, in W3C environments, the attempt to provide a standardized directory for service discovery resulted in the definition of the Universal Description, Discovery, and Integration (UDDI) specification, a platform-independent framework for publishing and discovering information about services [16]. OGC investigated the possibility of discovering the capabilities of its geospatial services through the UDDI interface by using SOAP messages. The experimental results of this attempt are reported in [17], where it is stated that UDDI is less suited both for obtaining the information to bind a service and for discovering specific contents or capabilities of individual service instances. Other important limitations include the fact that when using UDDI a client cannot "query a service by its interface signature", as well as its lack of support for metadata annotation and metadata-based service discovery [18].

A more flexible alternative to the use of public registries is the acquisition of metadata directly from the intended services. A client can directly interact with a service provider to get the desired information about available services. A requester only needs to know a priori the location where a provider offers metadata about its services and has to agree on a common protocol, the same as with the definition of the public interface and messaging system. This greatly simplifies the task, thus preventing clients from interacting, every time, with proprietary retrieval systems that are offered by services providers [2]. Flexibility with the typology of metadata that can be retrieved constitutes an additional desirable requirement. In a W3C-based environment, the standard way to directly retrieve

metadata about a service is by using the Web Services Metadata Exchange (WS-Metadata) specification that provides a standardized, SOAP-based, way for the encapsulation, insertion, retrieval, and removal of metadata associated with a W3C service [19]. Although WS-Metadata has not been expressly designed to deal with geospatial metadata in general and OGC metadata in particular, its underlying design choices make it a scalable solution, adaptable to arbitrary forms of metadata.

In order to better contextualize how our proposal impacts on the WS-Metadata protocol, a brief overview of the key-points of this W3C specification, and an analysis of the general structure of a generic OGC Capabilities document follow.

The WS-Metadata specification defines a bootstrap mechanism to get started with the actual metadata retrieval, the ability to support future versions of current metadata and the possibility to add further metadata formats. In its simplest form, a typical WS-Metadata message exchange pattern consists of a SOAP request that a requester sends to a service provider and a SOAP response that is sent back by the provider. In order to initialize such a communication, a client needs to know a priori the Service Endpoint, namely a location where the requester can send a SOAP message containing the request for the desired metadata (The discovery of an Endpoint address and the application of security policies to the bootstrapping phase are outside the scope of our discussion). The effective metadata encapsulation is achieved by the use of the <*Metadata*> Element, whose outline is shown in Figure 1 (In the remainder of the text, to improve the overall readability, when there is no ambiguity, we will omit the *mex* prefix. It is worth noting that as stated in the standard, the choice of the prefix is arbitrary and not semantically relevant). The <*Metadata*> Element can be seen as a container for one or more metadata units. Each metadata unit is represented by a <*MetadataSection*> and can be embedded into the <*MetadataSection*> Element or referenced through the <*MetadataReference*> or <*MetadataLocation*> Elements. The Dialect and Identifier attributes of a <*MetadataSection*> are mandatory. The former specifies the type and the version of the metadata embedded into the XML Element, the latter is an absolute Internationalized Resource Identifier (IRI) that identifies the specific metadata. The interpretation of the Identifier attribute is dialect-specific [19]. As for the actual metadata retrieval from a Service Endopoint, the current version of the WS-Metadata protocol defines two mechanisms, namely the GetWSDL operation, which is suitable to directly retrieve the WSDL document of a service, and the GetMetadata operation useful when the requester "whishes to obtain a specific metadata document" [19]. Figure 2 shows the typical structure of a GetMetadata request.

The optional Content attribute specifies the IRI for the request. When absent, the default value is "www.w3c.org/TR/ws-metadata-exchange". The <*Dialect*> Element is instead optional and deserves some additional considerations. According to the standard, all of the available metadata have to be returned when this Element is absent. However, when it is included in a <*GetMetadata*> request, the response has to return only those metadata matching the values specified by the combination of the mandatory Type and the optional Identifier and Content attributes. If no metadata is available for the specified combination, the response must not return any metadata for that Dialect element. Finally, if an Endpoint can accept a GetMetadata request, it has to send a GetMetadataResponse message, whose skeleton is reported in Figure 3.

As shown in the picture, for a GetMetadataResponse, the Body of the SOAP response message has to contain a <*Metadata*> Element.

Different from the W3C services, the three main OGC services (WMS, WFS, and WCS) do not provide flexibility and instead impose the exposure of metadata describing their capabilities. The only way for an OGC client to retrieve such metadata is through the invocation of the GetCapabilities operation whose aim is to allow "any client to retrieve metadata about the capabilities provided by any server that implements an OWS interface implementation specification" [20]. The typical response to a GetCapabilities request is an XML file, the Capabilities document, which contains metadata about the specific abilities of the invoked service. The structure of a Capabilities document is rigorously defined by OGC (although particular implementations "can provide additional operations returning service metadata" [20]) and should be ideally divided into two main parts, namely the aspects that are

common to all the OGC services and the sections that provide for metadata necessary for the specific functionalities of each single service type. Finally, it is also worth noting that OGC envisages that an XML encoded GetCapabilities request may be embedded in a SOAP message.

```
<mex:Metadata ...>
   <mex:MetadataSection
      Dialect='xs:QName'
      Identifier='xs:anyURI' ...>
      (
      <mex:MetadataReference ...>
      endpoint-reference-type
      </mex:MetadataReference>
      |
      <mex:MetadataLocation ...>
      xs:anyURI
      </mex:MetadataLocation>
      |
      DialectSpecificElement
      )
   </mex:MetadataSection> *
      xs:any*
</mex:Metadata>
```

Figure 1. The general structure of the <Metadata> Element.

```
<mex:GetMetadata Content='xs:any' ? ...>

   <mex:Dialect
      Type='mex:QNameSerialization'
      Identifier='xs:anyURI' ?
      Content='xs:anyURI' ? .../> *

      xs:any*

</mex:GetMetadata>
```

Figure 2. The <GetMetadata> Element.

```
<mex:GetMetadataResponse ...>

   <mex:Metadata ...>
      ...
   </mex:Metadata>

   xs:any*

</mex:GetMetadataResponse>
```

Figure 3. The <GetMetadataResponse> Element.

3. Extending the W3C Services Metadata Exchange Specification

As previously discussed, each OGC service offers by default the ability to retrieve all metadata useful for its profitable exploitation, while W3C services rely on additional protocols for the fulfillment of this task. However, when integrating OGC services in a W3C environment, forcing a SOAP-based client to directly query the OGC service and obtain the desired metadata is simply unfeasible for several reasons. In particular, this would cause an unnecessary additional complexity for the client, since it would require querying non SOAP-based entities and then process non-SOAP messages. Moreover, as this direct retrieval occurs outside the W3C service stack, another and probably more important issue refers to the incompatibility of such an option with the current SOAP-based standards (such as the Web Service Security [21]) meant to guarantee the security of information. Therefore, providing geospatial metadata in a W3C-compliant way is a challenge in order to support a better interoperability between these two proposals.

If the WS-Metadata specification directly supported the exchange of geospatial metadata, it would mean that a generic client could retrieve them during, for example, the bootstrapping phase, along with the information about the WSDL document. In fact, a wrapper could be configured as an Endpoint, and, on receiving a metadata request, transform the Capabilities document into a WS-Metadata compliant message and send it back to the requester. The actual Capabilities document could be retrieved by the wrapper either by directly querying the source OGC service or from a local cache. As previously stated, the specification has not been designed to directly support the exchange of non-W3C metadata, such as those that are related to OGC services. However, the WS-Metadata underlying design choices support future versions of known metadata formats, and, more importantly, allow for the addition of new formats. In this context, providing the WS-Metadata specification with the native support for OGC metadata essentially impacts on three main aspects:

1. the client has to be able to send to an Endpoint a GetMetadata message that explicitly refers to metadata of a particular OGC service;
2. the GetMetadata request has to discriminate among various types of OGC services (e.g., whether the request concerns the metadata for a WFS or a WMS); and,
3. the Endpoint has to support the request type and, in case it is unable to support the new metadata types, it has to simply reply with a fault message (as stated in the WS-Metadata specification).

Based on these considerations, the proposed approach to exchange OGC metadata through the WS-Metadata specification consists of extending the protocol by adding a new set of values combinations for the Type and Identifier attributes of the <*Dialect*> Element.

Two important constraints are to be taken into account. The former concerns the attribute format that has to be compliant with the WS-Metadata specification requirements. The latter requires that each combination has to uniquely refer to a specific OGC service. As for the Type attribute, it must be specified by a QName (Qualified name, a valid identifier for elements and attributes). Since the standard specifies that a QName must be serialized as "*{namespace-uri} localName*", the form "*{OGC service XML Namespace URI} Capabilities*" has been chosen, where the string between brackets represents the unique URI that is used in the namespace declaration of each type of OGC service. Finally, for the Identifier attribute, the actual URL of the XML Schema has been chosen that, for each type and version of OGC service, defines the structure of every admissible Element and its attributes. As for the third <*Dialect*> attribute, i.e., the Content, the default value defined in the specification has been used, namely www.w3c.org/TR/ws-metadata-exchange. The combination of values for the current versions of the three main OGC standards is shown in Table 1.

In order to be effectively exploitable in a real context, the proposed protocol extensions require the support of the underlying service infrastructure that has to properly manage the new types of requests and responses. In the next Section, a concrete example is discussed that shows how the proposed changes to the WS-Metadata protocol impact on the official Web services stack of the Java Enterprise Edition (Java EE, for short) platform.

Table 1. New values for the Type and Identifier attributes to support the three main Open Geospatial Consortium (OGC) standards.

OGC Service Type	Type Attribute Value	Identifier Attribute Value	Content Attribute Value
WFS	{*http://www.opengis.net/wfs*} *Capabilities*	*http://schemas.opengis.net/wfs/2.0/wfs.xsd*	*www.w3c.org/TR/ws-metadata-exchange*
WMS	{*http://www.opengis.net/wms*} *Capabilities*	*http://schemas.opengis.net/wms/1.3.0/capabilities_1_3_0.xsd*	*www.w3c.org/TR/ws-metadata-exchange*
WCS	{*http://www.opengis.net/wcs*} *Capabilities*	*http://schemas.opengis.net/wcs/2.0/wcsGetCapabilities.xsd*	*www.w3c.org/TR/ws-metadata-exchange*

As for the application of the proposed approach and an example of architecture, the authors are involved in an international project, named Social Life Networks for the Middle of the Pyramid (http://www.sln4mop.org/sln/home), where a large amount of data exchange is required. Data is heterogeneous and complex and is distributed over different platforms. Some of them have been specifically built for the project, others are derived and integrated from existing domains. The wrapper-based solution has been initially investigated to address that issue, namely to allow for data exchange among the different infrastructures that are involved in the project. In [22], a preliminary version of the architecture prototype underlying a digital knowledge ecosystem is described. It allows for producing knowledge through aggregating and mining processes that are applied on data exchanged among different infrastructures.

4. Extending the Java EE Web Services Stack to Support the Direct OGC Metadata Retrieval

In the Java EE platform, the Java API for XML Web Services (JAX-WS) specification [23] represents the default approach to develop service-based solutions that are established on the SOAP and WSDL standards. From a server side point of view, the operations exposed into a WSDL document are mapped into traditional methods of Java classes. On the client side, a proxy (i.e., an object representing the intended service) is created and a client simply invokes its methods. The JAX-WS runtime then converts requests and responses into the corresponding SOAP messages. Although the JAX-WS programming model can be implemented by any software vendor, the present investigation focused on its reference implementation, the JAX-WS RI [24]. The JAX-WS RI constitutes the core layer of Metro [25], which is an open source project representing the official Web service stack of the Java EE platform. The other fundamental layer of the Metro stack is represented by the Web Services Interoperability Technologies (WSIT) subsystem. WSIT is built on top of JAX-WS RI and provides for the concrete implementation of several additional Web service specifications dealing with enterprise-level features such as reliability, security and transactions. Moreover, in order to foster interoperability among service-based solutions that were developed with different software technologies (e.g., the Microsoft's.NET framework), WSIT offers a bootstrapping mechanism that supports the actual retrieval of the service's WSDL document and metadata. Such a functionality extensively relies on the WS-Metadata protocol.

To better contextualize the scope of our changes, in the next subsection, we first provide a high-level overview of the global structure of WS-Metadata module available in WSIT, and then briefly describe the role that is played by the most important Java classes that actually support the metadata exchange process.

4.1. The WSIT Metadata Exchange Module

From an implementation point of view, the entire Metro project is made up of several Java packages that handle the core W3C standards as well as the WS-* additional specifications. In particular, the actual implementation of WS-Metadata protocol is organized into four packages: client, server, mex, and client.schema. These packages group all the source classes according to their role in the whole metadata exchange process. The first two packages, client and server, contain the classes that let Java-based Web services clients exchange metadata with third party WS-Metadata enabled solutions.

The mex package contains only the MetadataConstants class, whose aim is to store several useful fixed strings, such as the supported versions of the SOAP and WSDL specifications and the prefix of the mandatory XML namespaces. Finally, the client.schema package encloses the Java classes that map the fundamental components of the WS-Metadata specification, namely the <*Metadata*>, <*Metadata Section*>, <*Metadata Reference*>, and <*Get Metadata*> Elements.

The core component of the client package is the MetadataClient class whose main aim is to provide developers with a convenient way to handle XML-based metadata elements as Java object as well as to obtain additional service information, such as its port QNames. The most important part of this class is the retrieveMetadata() method that performs the two-step process of retrieving a metadata set from a service Endpoint and convert it into Java instances. The retrieval phase is implemented, as follows. The method initially attempts to make a request using the SOAP 1.2 protocol. If such a version is not supported by the service Endpoint, it retries using Version 1.1. In case both attempts fail, it tries to retrieve metadata by adding the mex prefix to the Internet address of the service. For actual metadata retrieval, retrieveMetadata() internally invokes the GetMetadata() method from the MetadataUtil class. The purpose of this utility method is to send the WS-Metadata request to a server. To accomplish its task, it invokes in turn the getMexWsdlRequest() method, which builds the SOAP message containing the request. Once the retrieveMetadata() obtains a full response from the service, it performs the second process step, by invoking the createMetadata() method that basically removes the metadata from the SOAP message and returns a Metadata object.

The source class of the Metadata object is contained in the client.schema package. During the metadata retrieval phase, the above-mentioned methods take advantage of the functionalities that are provided by several utility classes, including the HttpPoster and the PortInfo classes. The former performs the task of making an HTTP POST request to a service; the latter holds information, such as the service name, the qualified name of its port, and the port address. Finally the ServiceDescriptorImpl class is a utility class that can be invoked by the underlying JAX-WS layer to access and use service metadata from an Endpoint.

As for the various server side components, the MexEndpoint class is of particular importance for the goal of the present proposal. As the name suggests, this class acts as an Endpoint entry for a Web service. The class implements the invoke() method of the Provider interface, introduced in the JAX-WS specification to provide for a more fine-grained control over XML-based messages. The main goal of the invoke() implementation available in the Mex-Endpoint class is to act as a dispatcher that invokes additional auxiliary methods, according to the request type. In particular, to differentiate among such requests, it parses the value of the Action Element available in the Header field of a SOAP request message. In this specific case the processGetRequest() method is invoked. By exploiting the WSDL Retriever class, this method first obtains the requested WSDL and then writes it in the Body of a SOAP response Message. The source code of the above-mentioned methods has constituted the scaffolding to provide a service Endpoint with both the support for GetMetadata requests, as well as the ability to manage geospatial metadata.

4.2. Adding the OGC Metadata Support to the WSIT Metadata Exchange Module

In this section, we discuss technical details of our proposed modification of the W3C WSIT Metadata Exchange module to add support for OGC metadata. As previously mentioned, each implementation that aims at altering W3C standards in support of geospatial metadata should be totally transparent for existing clients and services that do not deal with geospatial information. To achieve this aim we left, whenever possible, the actual methods implementation and semantic of the original WSIT classes unaltered. Our solution to support OGC metadata has been realized by adding new methods that are specifically designed to manage geospatial metadata requests and responses to the involved classes. The code snippets provided in this section use the retrieval of the WFS Capabilities as example. Of course, the whole discussion can be easily extended to every OGC service that can provide its metadata using a standard-compliant Capability document.

The overall modifications were, however, influenced by several aspects. First of all, the WSIT subsystem lacks the support for the latest version of the WS-Metadata specification. In fact, according to the information that is available in the source code, the supported specification is the WS-Metadata 1.1, dated September 2004. Nevertheless, this issue did not have a notable impact on the purposes of our discussion since the semantic and behavior of the <*Metadata*>, <*MetadataSection*> and <*GetMetadata*> Elements and the Dialect and Identifier attributes do not differ from the brief description that is provided in Section 3. Secondly, to accelerate the development process, an initial simplification has been made. A requesting client knows a priori that the receiving service supports the geospatial metadata request and the associated dialect. In a real context, such information could be retrieved, for example, from a service registry or the provided documentation. Finally, the lack of an official documentation for several WSIT modules has considerably complicated the whole design and development process. In the next two subsections we describe the main changes that are required to the original WS-Metadata modules to let both client and service deal with the new metadata types.

4.2.1. Adding the OGC Metadata Support to the WSIT Metadata Exchange Module–Client Changes

In order to provide a generic Java application with the ability of both retrieving the capabilities of an OGC compliant WFS, properly understanding the service response, and effectively using the retrieved metadata, the following changes to the client package classes are required. First of all, a unique way to address the new Dialect inside the Java source code is necessary, then a new CAPABILITIES DIALECT string uniquely identifying the WFS dialect has been added to the list of constants that are included in the MetadataConstants class, as follows:

public static final String CAPABILITIES_DIALECT = "http://schemas.opengis.net/wfs/2.0/wfs.xsd";

As for the actual retrieval and use of the WFS capabilities, a desirable requirement was to keep the semantic of the original two-step process, namely getting the service metadata and instantiating related Java objects, unaltered. An advantage of this choice was the ability to entirely reuse the createMetadata() method to return Metadata objects, thus allowing for focusing only on the changes necessary for performing a GetMetadata request through the new Dialect. To accomplish this task, the main classes of the client package have been modified by adding several methods meant to both seamlessly handle the retrieval of the new metadata types and overcome some minor restrictions of the WSIT original implementation. Figure 4 shows the interactions sequence occurring during the retrieval process, while Table 2 shows the classes affected by the above changes.

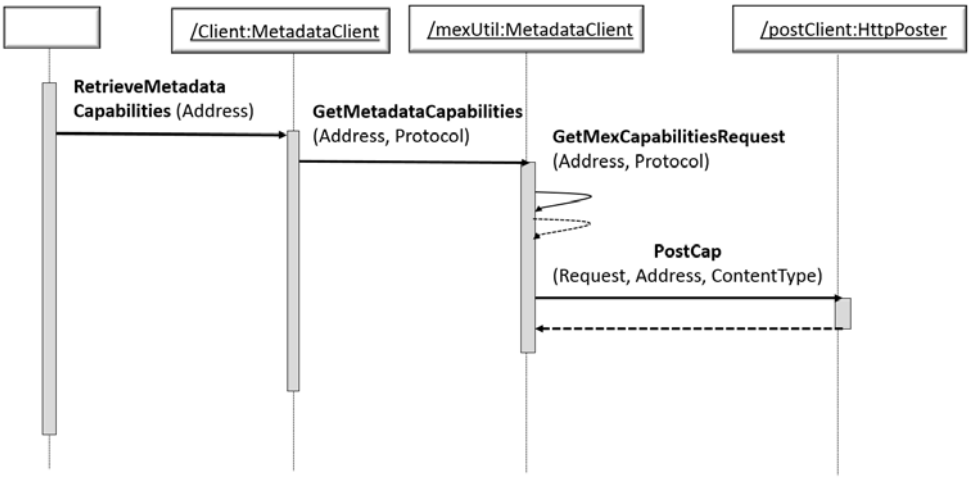

Figure 4. Classes and methods involved in the metadata retrieval phase.

Table 2. Java classes of the client package affected by the changes.

Class Name	Methods Added or Modified
MetadataClient	retrieveMetadataCapabilities()
MetadataUtil	getMetadataCapabilities(); getMexCapabilitiesRequest()
HttpPoster	postCap()
ServiceDescriptorImp	Class constructor; handleXML(); handleLocation(); getCapabilities()

The invocation of the retrieveMetadataCapabilities() method is the first step that a Java client performs to begin the geospatial metadata exchange process. The method body implements the above-mentioned two-step process of getting metadata and creating the corresponding Java objects. In particular, to carry out the first task an additional method, getMetadata-Capabilities(), in the MetadataUtil class is invoked. The goal of getMetadataCapabilities() is to return an InputStream object that encodes the XML stream that was retrieved from the service Endpoint. The goal is fulfilled with the in-sequence invocation of two other methods, namely getMexCapabilitiesRequest() and postCap(), from the HttpPoster class. The purpose of the first method is to build the SOAP message to be sent to the service Endpoint. The two parameters that are needed are the Web service address along with its mex suffix and the version of the SOAP protocol used. Finally, the postCap() invocation makes the real HTTP POST request to the service Endpoint. It is worth noting that we needed to slightly modify the method source code to provide the HTTP request with the ability to properly manage a GetMetadata. Such a feature was not available in the original implementation.

4.2.2. Adding the OGC Metadata Support to the WSIT Metadata Exchange Module–Server Changes

Providing a service Endpoint with the ability to correctly manage the SOAP request message and to create a feasible response (or a proper fault message) constituted the main theme that led the design of the changes made to the server package in general and to the MexEndpoint class in particular. Moreover, as done for the client package during its actual implementation, the goal was pursued of keeping unchanged the overall business logic and to reuse the functionality that is available in the original package implementation.

In order to be effective, from a logical point of view, such modifications can be grouped into three main categories, although they are tightly tied each other, namely:

1. overcoming the lack of support for GET METADATA requests inside the invoke() method,
2. making all of the involved classes and methods aware of the new metadata dialects, and
3. enabling the service Endpoint to seamlessly access and parse the original Capabilities documents supplied by remote OGC-compliant services and encapsulate their content into a canonical MetadataSection Element.

As for the first task, on receiving a GET METADATA request, the original implementation of the invoke() method simply returns a fault message (GET METADATA NOT IMPLEMENTED). Therefore, the initial step in our approach concerned the need to modify the method's source code by adding the support to properly manage and dispatch such a type of requests (Figure 5).

The Action field that is embedded in the Header Element of a SOAP message constitutes the discriminating factor used by the invoke() native implementation to distinguish among valid metadata request types. Hence, on receiving a well-formed metadata request, the current value of the Action field and the value of the GET METADATA REQUEST constant stored into the MetadataConstants class are compared. If the content of the action object and the above-mentioned constant are equivalent, the processGetMetadataRequest() method is invoked (Figure 6). The goal of the method is to construct the response message containing the geospatial metadata requested by the client. As mentioned, the fulfillment of this task requires the execution of two different activities. First of all, it is essential to access and parse the Capabilities document containing the intended metadata and return its Java-based

representation. Then, an automated procedure that performs all of the low-level steps to correctly embed the retrieved information into a SOAP response message is required. Such issues are faced by improving functionalities of the server package through two new classes, namely CapabilitiesRetriever and CapabilitiesUtility, containing several utility methods (Table 3).

```
1   /.............../
2
3   String action = headers.getAction(AddressingVersion.W3C,
        soapVersion);
4
5   /.............../
6
7   else if (action.equals(GET_REQUEST)) {
8       final String toAddress = headers.getTo(wsaVersion,
            soapVersion);
9       return processGetRequest(requestMsg, toAddress,
            wsaVersion, soapVersion);
10  }
```

Figure 5. Changes to the invoke method.

Table 3. Additional classes added to the server package.

Class Added	Methods
CapabilitiesRetriever	addDocumets(), writeDoc()
CapabilitiesUtility	writeDocTo()

```
1   private Message processGetMetadataRequest(final Message request,
2       String address, final AddressingVersion wsaVersion,
3       final SOAPVersion soapVersion) {
4
5       try {
6
7           WSEndpoint ownerEndpoint = findEndpoint();
8
9           if (ownerEndpoint != null) {
10
11              final MutableXMLStreamBuffer buffer = new
                    MutableXMLStreamBuffer(1024000);
12              final XMLStreamWriter writer =
13
14              address = this.getAddressFromMexAddress(address,
                    soapVersion);
15              writeStartEnvelope(writer, wsaVersion, soapVersion);
16              CapabilitiesRetriver cr = new
                    CapabilitiesRetriver(ownerEndpoint);
17              cr.addDocuments(writer, null, address);
18              writeEndEnvelope(writer);
19              writer.flush();
20
21              /....... OTHER CODE ........./
```

Figure 6. The processGetMetadataRequest method.

The addDocuments() method retrieves a local copy of the Capabilities document from a predefined disk location. Such a byte stream constitutes, in turn, one of the mandatory inputs for the following invocation of the writeDoc() method. The invoked method builds the MetadataSection element that will contain the Capabilities document that was obtained in the previous step. As described in Section 3, in addition to the actual content, MetadataSections Elements have to contain the Dialect and Identifier attributes of the metadata unit. To accomplish this task, the CAPABILITIES DIALECT string can be reused. Finally, to improve the code reusability and maintainability, the Java code dealing with the addition of the Capabilities documents content to the payload of a MetadataSection Element has been separated from the remaining part of the methods body. This specific task is accomplished by the writeDocTo() method (Figure 7).

A critical aspect of such an operation that it is worth to analyze is represented by the binding process of GML objects into Java classes. The direct mapping from OGC Schemas into usable Java objects is, in fact, not a straightforward process. In the Java platform, one of the most common ways to access and manage XML documents is by using the Java Architecture for XML Binding (JAXB) API [26], a specification whose main goal is to simplify the integration of XML and Java technology by providing an object-oriented representation of the XML documents. According to the official documentation, the actual use of this API can be seen as a two-step process:

1. Binding the Schema of an XML document into a series of Java interfaces and classes representing the Schema elements. Each JAXB compliant implementation provides a binding compiler, i.e., a tool that automatically performs this task.
2. Unmarshalling of the XML document: creating an objects tree that represents the actual content and structure of the XML document.

The objects in the tree are instances of the classes that are produced in the binding step. During the unmarshalling phase, it is also possible to validate the source data against the target XML Schema. However, the proper use of the JAXB API (in particular, the use of XJC, the binding compiler of JAXB Reference Implementation) with the OGC XML Schemas represents a challenging task. Major issues to solve include the proliferation of versions of several Schemas, Schemas that are not valid (e.g., GML Version 3) or depend on several others Schemas. Such a complexity makes an OGC Schema difficult to be directly compiled as is. In this context, a support that dramatically simplifies the Java-based implementation of the OGC specifications can be found in the bindings and libraries provided by the OGC Schemas and Tools Project whose aim is to "compile all of the OGC XML Schemas with JAXB schema compiler" [27]. According to the documentation, the project supports the 1.0.0 and 1.1.0 Versions of the WFS Schemas (the latter being the same as the latest 2.0 version) and the latest versions of the 3.X.X branch of GML (from 3.1.1 to 3.2.1). In this context, the core elements in Listing 4.3 are the JAXBContext, the ObjectFactory, the WFSCapabilitiesType, and the Marshaller classes. The general role of the first class is to provide a customized binding that will provide the necessary information for the marshalling, unamarshalling, and validation phases. The ObjectFactory class contains factory methods for each Java element that has been generated starting from the *net.opengis.wfs.v_1_1_0* package. The WFSCapabilitiesType represents the Java class that corresponds to the WFS CapabilitiesType complex type, namely the XML Element defined in the WFS Schema that specifies the structure of the service metadata. Finally, with the Marshaller class the in-memory Java objects are converted into the XML file that will constitute the Capabilities document.

This section shows the technical details of a working example of how our proposed changes to the WS-Metadata protocol affect the official Web services stack of the Java EE platform. This concretely illustrates the viability of our solution. Advantages and benefits of our approach are discussed in the next section with a view to further developments.

```
public void writeDocTo(File file, final XMLStreamWriter writer,
    final String address) {
        try {
            JAXBContext context =
                JAXBContext.newInstance("net.opengis.wfs.v_1_1_0");

            Unmarshaller unmarshaller = context.createUnmarshaller();
            ObjectFactory wfs_factory = new ObjectFactory();

            JAXBElement<WFSCapabilitiesType> wfscapabilitiesElement
                = (JAXBElement<WFSCapabilitiesType>)
             unmarshaller.unmarshal(new StreamSource(file),
                 (wfs_factory.createWFSCapabilitiesType()).getClass());

            WFSCapabilitiesType capabilitiesElement =
                wfscapabilitiesElement.getValue();

            Marshaller jaxbMarshaller = context.createMarshaller();
            jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
                Boolean.TRUE);
            jaxbMarshaller.marshal
            (wfs_factory.createWFSCapabilities(capabilitiesElement),
                writer);


        } catch (JAXBException ex) {
            Logger.getLogger
            (CapabilitiesUtility.class.getName()).log(Level.SEVERE,
                null, ex);
    }
}
```

Figure 7. The writeDocTo method.

5. Conclusions and Future Works

When services that are developed according to different sets of standards are involved in a composition process, several interoperability issues may arise that cannot be fixed by a mere mechanical process and require ad-hoc solutions. A solution is represented by the development of a wrapper that abstracts the inner details and the logic of the wrapped service as much as possible, thus embracing the low-coupled nature of the entire SOC paradigm. Moreover, the adherence to accepted standards and the ability to reuse the implemented functionality are basic requirements for such a type of software solution.

As for the specific case of OGC and W3C services, the development of a wrapper that exposes OGC functionality in a W3C-compliant way represents a viable option to overcome the technical differences between the two types of services. However, the wrapper development process is not a straightforward task and a complete mapping from the OGC standards to the W3C ones is still an open research question.

In this paper, issues that are related to the proper management of geospatial metadata into a W3C-based environment are specifically addressed. Due to the fundamental role of metadata for a proper exploitation of the information conveyed by an OGC service, it was essential to provide an adequate mechanism for discovering and using them. The accomplishment of this task has been based on a flexible and standard-compliant mechanism for retrieving such metadata directly from a wrapper, thus avoiding the use of third-party entities like UDDI registries.

The main advantages of the present scientific contribution can be summarized, as follows:

1. a tight adherence to the syntactic and semantic requirements of the WS-Metadata protocol;
2. the ability to seamlessly support different types of OGC-compliant services along with different versions of the same OGC service; and,
3. the capability to easily reuse the proposed functionality in a wider development context since the proposed theoretical approach has been concretely implemented as an extension of an official module of the Java Enterprise Platform.

Additional direct consequences of the entire design approach include the reduced effort that is required to adapt the developed module to other types of services that exchange their metadata using XML-based formats and its strong integration with other features of existing service middleware.

Notwithstanding the advantages that are provided by this solution, there are still some limitations. Indeed, a wrapper typically performs a one-to-one mapping with an intended OGC service i.e., the translation of request and response messages is explicitly tailored to the specific characteristics of the wrapped service. This limits the capability to adapt a solution that is designed for a certain OGC service to another one. To overcome such a limitation and to minimize the required changes, it is necessary to further investigate and enhance two distinguishing features of a wrapper, namely the retrieval phase of a Capabilities Document and the description of the public interface using the WSDL standard. Based on these considerations, the focus of a future investigation will be mainly on improving the wrapper flexibility and reusability. In particular, the OGC Common Standard that strictly regulates the implementation of the GetCapabilities operation envisages the ability for an OGC client to request only certain sections of the service metadata document. Although this is an optional functionality, providing support for such a feature would further increase the flexibility of the WS-Metadata protocol, thus simplifying the development of a one-to-many wrapper.

Author Contributions: All the authors contributed to the design and implementation of the research, to the analysis of the results and to the writing of the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Papazoglou, M.P. Service oriented computing: Concepts characteristics and directions. In Proceedings of the Fourth International Conference Web Information Systems Engineering, Rome, Italy, 10–12 December 2003; pp. 3–12. [CrossRef]
2. Erl, T. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*; Prentice Hall: Upper Saddle River, NJ, USA, 2005.
3. Tsalgatidou, A.; Pilioura, T. An overview of standards and related technology in web services. *Distrib. Parallel Databases* **2002**, *12*, 135–162. [CrossRef]
4. Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S. *Web Services Description Language (WSDL) 1.1*; World Wide Web Consortium: Cambridge, MA, USA, 2001.
5. Gudgin, M.; Hadley, M.; Mendelsohn, N.; Moreau, J.J.; Nielsen, H.F.; Karmarkar, A.; Lafon, Y. *Soap Version 1.2*; World Wide Web Consortium: Cambridge, MA, USA, 2001.
6. Zhao, P.; Yu, G.; Di, L. *Geospatial Web Services. Emerging Spatial Information Systems and Applications*; Idea Group Publishing: Hershey, PA, USA, 2007; pp. 1–35.

7. Villa, M.; Di Matteo, G.; Lucchi, R.; Millot, M.; Kanellopoulos, I. *Inspire Network Services SOAP Framework*; JRC Scientific and Technical Reports; European Commission, Joint Research Centre, Institute for Environment and Sustainability: Ispra, Italy, 2008.
8. Alves, A.; Arkin, A.; Askary, S.; Barreto, C.; Bloch, B.; Curbera, F.; Ford, M.; Goland, Y.; Guzar, A.; Kartha, N.; et al. *Web Services Business Process Execution Language Version 2.0*; OASIS Standard; OASIS: Burlington, MA, USA, 2007.
9. Ioup, E.; Lin, B.; Sample, J.; Shaw, K.; Rabemanansoa, A.; Reimbold, J. Geospatial web services: Bridging the gap between OGC and Web services. In *Geospatial Services and Application for the Internet*; Springer: New York, NY, USA, 2008; pp. 73–93.
10. Amirian, P.; Alesheikh, A.A.; Bassiri, A. Standards-based, interoperable services for accessing urban services data for the city of Teheran. *Comput. Environ. Urban Syst.* **2010**, *34*, 309–321. [CrossRef]
11. Sancho-Jiménez, G.; Béjar, R.; Latre, M.; Muro-Medrano, P.R. A method to derivate SOAP interfaces and WSDL metadata from the OGC Web processing service mandatory interfaces. In *Advances in Conceptual Modeling: Challenges and Opportunities*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 375–384.
12. Bertolotto, M.; Di Giovanni, P.; Sebillo, M.; Tortora, G.; Vitiello, G. The information technology in support of everyday activities: Challenges and opportunities of the service oriented computing. *Mondo Digit.* **2014**, *49*, 1–12.
13. Di Giovanni, P.; Bertolotto, M.; Vitiello, G.; Sebillo, M. Web Services Composition and Geographic Information. In *Geographical Information Systems: Trends and Technologies*; CRC Press: Boca Raton, FL, USA, 2014; pp. 104–141.
14. Nogueras-Iso, J.; Zarazaga-Soria, F.J.; Béjar, R.; Álvarez, P.J.; Muro-Medrano, P.R. OGC Catalog Services: A key element for the development of Spatial Data Infrastructures. *Comput. Geosci.* **2005**, *31*, 199–209. [CrossRef]
15. OpenGIS Catalogue Service. *OpenGIS Catalogue Service Implementation Specification (ISO 19115), v2.0.2. CSW 2.0.2*; Open Geospatial Consortium Inc.: Wayland, MA, USA, 2007.
16. Clement, L. *UDDI Version 3.0.2*; OASIS Standard; OASIS: Burlington, MA, USA, 2005.
17. Lieberman, J.; Reich, L.; Vretanos, P. *Ows 1.2 UDDI Experiment*; Open Geospatial Consortium Inc.: Wayland, MA, USA, 2003.
18. Fang, W.; Moreau, L.; Ananthakrishnan, R.; Wilde, M.; Foster, I. Exposing UDDI service descriptions and their metadata annotations as WS-resources. In Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, 28–29 September 2006; pp. 128–135. [CrossRef]
19. Davis, D.; Malhotra, A.; Warr, K.; Chou, W. *Web Services Metadata Exchange (WS-MetadataExchange) W3C Recommendation*; World Wide Web Consortium: Cambridge, MA, USA, 2011.
20. Whiteside, A.; Greenwood, J. *Ogc Web Services Common Standard*; Open Geospatial Consortium Inc.: Wayland, MA, USA, 2010.
21. Nadalin, A.; Kaler, C.; Monzillo, R.; Hallam-Baker, P. *Web Services Security: Soap Message Security 1.1 (WS-Security 2004)*; OASIS Standard; OASIS: Burlington, MA, USA, 2004.
22. Sebillo, M.; Vitiello, G.; Ginige, A. "Creating Territorial Intelligence through a Digital Knowledge Ecosystem: A Way to Actualize Farmer Empowerment". In Proceedings of the 17th International Conference on Computational Science and Its Applications (ICCSA 2017), Trieste, Italy, 3–6 July 2017; pp. 98–111. [CrossRef]
23. Kotamraju, J. *The Java API for XML-Based Web Services (JAX-WS) 2.2*; Sun Microsystems, Inc.: Santa Clara, CA, USA, 2009. Available online: http://download.oracle.com/otn-pub/jcp/jaxws-2.2-mrel3-evalu-oth-JSpec/jaxws-2_2-mrel3-spec.pdf?AuthParam=1458484690_4827e9622d437f94a2c141930ca9b347 (accessed on 21 March 2016).
24. JAX-WS RI Project. Available online: https://jax-ws.java.net/ (accessed on 21 March 2016).
25. Metro Project. Available online: https://metro.java.net/ (accessed on 21 March 2016).
26. Ort, E.; Mehta, B. *Java Architecture for XML Binding (JAXB)*; Sun Developer Network; ORACLE: Redwood City, CA, USA, 2003.
27. OGC Schemas and Tools Project. Available online: http://www.ogcnetwork.net/jaxb4ogc (accessed on 21 March 2016).