International Journal of Digital Earth,
Vol. 2, Supplement 1, 2009, 40–61

RESTful implementation of geospatial services for Earth and Space Science applications

P. Mazzetti[a,b]*, S. Nativi[a,b] and J. Caron[c]

[a]*Institute of Methodologies for Environmental Analysis (IMAA), Italian National Research Council (CNR), Tito Scalo, Italy;* [b]*PIN, University of Florence, Prato, Italy;* [c]*UNIDATA-UCAR P.O. Box 3000 Boulder, CO 80307-3000, USA*

(Received 12 January 2009; final version received 7 February 2009)

In recent years, Representational State Transfer (REST) has been proposed as the architectural style for the World Wide Web. REST promises of scalability and simple deployment of Web Services seem to be particularly appealing for Earth and Space Science (ESS) applications. In fact, most of the available solutions for geospatial data sharing, applying standard interoperability specifications, require complex service-oriented infrastructures; these are powerful and extensible environments, but they usually result in difficult to deploy and manage for ESS research teams. Thus, ESS researchers would gain great benefit from an easy way of sharing geo-information using the international interoperability standards. The variety and complexity of geo-information sharing services poses several architectural issues; in fact these services encompass sensor planning and observation, coverages and features publication and retrieving, models and simulations running, data citation and annotation. Consequently, the adoption of a specific architectural style must be carefully evaluated against these specific requirements. In this work we analyse the existing geospatial services from an architectural perspective and investigate their possible RESTful implementation. Particular attention is paid to the OGC Web Coverage Service (WCS). Possible benefits and drawbacks, along with open issues and possible solutions are discussed. Our investigation suggests that REST may fit well to the typical ESS research usage cases. However, the architectural choice (e.g. Simple Object Access Protocol (SOAP) vs REST) will depend on a case-by-case analysis. Other important factors must be considered, such as the application context: a valuable example in point are the e-Business and e-Government application scenarios which require message based solutions – like those implemented by SOAP. In any case, there is a clear need for harmonization and reconciliation of these two approaches.

Keywords: geospatial web services; ROA; REST; WCS; SOA; e-Science

Introduction

Earth and Space Science Informatics (ESSI) is a recent discipline aiming to provide scientists with advanced information and computational services in support of Earth and space systems research. In such a context, geospatial services play an important role enabling geo-information sharing; this is essential to provide scientists with services for data discovery, publishing and access. Besides, the geospatial services are

*Corresponding author. Email: mazzetti@imaa.cnr.it

ISSN 1753-8947 print/ISSN 1753-8955 online
© 2009 Taylor & Francis
DOI: 10.1080/17538940902866153
http://www.informaworld.com

the building blocks for more complex disciplinary services – e.g. processing, simulation, etc.

Since its beginnings the World Wide Web (WWW) was chosen as the preferred infrastructure for geospatial services. Most initiatives for the specification and standardization of geo-information resources (e.g. services, models, and formats) adopted Web technologies as their protocols and encodings; also, they used the Web interaction model for services based on the navigation paradigm. A significant case in point is the Open Geospatial Consortium (OGC) which converted its specifications to Web technologies from other solutions – e.g. CORBA (http://www.corba.org/), OLE/DCOM (http://www.microsoft.com/COM/) – since the end of the 1990s. The adoption of WWW technologies allowed to benefit from the pervasiveness and scalability of the Web and from the dynamicity of its development community which provides an ever-growing set of open specifications and solutions. Therefore, today Web technologies, and in particular HTTP, Simple Object Access Protocol (SOAP), and XML (http://www.w3.org/XML/) are the cornerstones for designing and developing really interoperable geospatial services.

In recent years the WWW has undergone important changes. The advent of new technologies – e.g. AJAX (http://www.w3schools.com/ajax/default.asp), JSON (http://www.json.org/) – new services – e.g. Web 2.0 services (O'Reilly 2005) – and new architectural approaches – e.g. Representational State Transfer (REST) (Fielding 2000) – although often based on newly interpreted existing solutions, have deeply changed the way the Web is built and experienced by the users (O'Reilly 2005). Since REST has been proposed as the architectural style for the Web, it promises the scalability of the original Web and the simplicity for services development and deployment. While, the traditional Web and the powerful, but complex, Service-Oriented-Architectures (SOA) do not seem to provide these benefits. In the Web community, a great debate arose about the real and the 'best' architecture for the future Web; essentially, each approach has its own advantages and disadvantages. Therefore, the architecture choice should be actually based on the requirements emerging from specific usage cases. In keeping with that, REST architectures and related technologies seem to be appealing when simplicity and flexibility are the main requirements as it is common for research applications. Indeed, scientists' main objective is not to develop and maintain the complex infrastructures required for SOA implementation, but to access and publish information in the easiest possible way. Thus, geospatial services for the Earth system science might gain benefit from REST architectures; an accurate investigation and experimentation is needed.

Service-oriented and resource-oriented architectures

The characteristics of a distributed system like the WWW are determined by its architecture, which could be defined as '*the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution*' (ANSI/IEEE 2000). Actually, many important characteristics can be inferred from the architectural style instead of the particular architectural instance. In fact, since an architecture embodies both functional and non-functional properties, it may be difficult to directly compare architectures conceived for different types of systems. Styles are a mechanism for categorizing architectures and for defining their common characteristics. An architectural style can be defined as '*a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style*' (Fielding 2000).

The architectures adopted in the development of Web systems can be classified in three main categories: Object-Oriented Architectures (OOA), SOA and Resource-Oriented Architectures (ROA). The former derives from the Object-Oriented approach, successfully used for the design and development of computational systems. It was proposed also for the Web through the development of specifications for remote method invocations – e.g. CORBA, Java RMI (http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp), XML-RPC (http://www.xmlrpc.com/), SOAP-RPC (http://www.w3.org/TR/2003/REC-soap12-part2-20030624/#soapforrpc) – on top of Web and Internet protocols. Several drawbacks limit the usefulness of OOAs in a heterogeneous environment as the Web is; they are mainly related to the strict coupling between the interacting objects (Vogel 2003). Hence, the other two approaches, SOA and ROA, are now the sole competitors in the Web arena. Indeed both SOA and ROA provide means for loosely-coupled access to the logical resources involved in the provision of a service. The main difference between these two approaches is that SOAs define (and limit) the way the interaction can be performed, while ROAs allow a basic but direct interaction with the resources.

Service-oriented-architectures (SOA)

SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. SOA is a means of organizing solutions that promote reuse, growth and interoperability. It is not itself a solution to domain problems but rather an organizing and delivery paradigm that enables one to get more value using capabilities which are locally 'owned' and those under the others control. SOA reflects the reality that ownership boundaries are a motivating consideration in the architecture and systems design.

The central focus of SOA is the task or business function – getting something done. Indeed, the central concept of SOA is the service: a mechanism to enable access to a set of one or more capabilities (OASIS 2006). A service can enable users to perform arbitrarily complex tasks involving the resources which are handled by the service provider and not directly exposed to the user. Therefore, SOA defines a class of architectures which enable loosely-coupled access to generic capabilities provided by service providers (Thomas 2005). The generality of services in terms of information, structure, semantics, behavior, action and process models require the provision of functionalities supporting visibility and awareness – through service description and policy definition. This makes SOA really powerful but complex, especially if only simple tasks are required.

Resource-oriented architectures (ROA)

ROA is a paradigm for organizing and utilizing distributed resources that may be under the control of different ownership domains. Unlike SOA, the ROA central focus is the resource – a logical entity which is exposed for direct interaction

(Overdick 2007). In SOA the human or machine user interacts with a distributed system through delegation, that is specifying the desired actions to a computational component instead of directly acting on the resources. For instance, the user can perform a complex task like 'generate model forecasts' delegating actions to the information systems (through a service invocation) instead of acting directly on the involved logical resources like the 'forecast model', 'input data', 'output data' and so on. On the other side, in ROA the human or machine user interacts directly with the exposed resources. Table 1 reports the main characteristics of both SOA and ROA approach.

SOA and ROA in the web

In principle, the Web is an information system characterized by a small set of open specifications – at the minimum the URI addressing standard, only. Thus, the Web could accommodate both architectural approaches. This means that a Web system could be based either on a SOA or a ROA. In recent years both approaches have been proposed for Web Systems and also as the basis for the Web itself. In fact, to assure the preservation of Web capabilities and characteristics a clear architecture definition is needed.

SOAP specifications suite

In the WWW the emergence of SOAP provided a common specification for service invocation between Web components. Lately SOAP, originally born for conveying remote methods invocation in XML, being fully suitable for generic messaging between objects, evolved in a more general standard for sending services calls targeted to endpoints exposed in the Web and addressed through a specific URL (Box 2001). Further specifications such as WSDL (http://www.w3.org/TR/wsdl), UDDI (http://www.oasis-open.org/committees/uddi-spec/), WS-I (http://www.ws-i.org/), WS-*, etc. from various standardization bodies, mainly W3C (http://

Table 1. SOA and ROA main characteristics.

	ROA approach	SOA approach
Web architectural style	REST	W3C Web services (SOAP Framework)
Web services implementation	RESTful (HTTP, XML, ...)	SOAP suite (SOAP, WSDL, ...)
Key concept	Resource (exposed logical entity)	Service (mechanism to access capabilities)
Identification	Resource address	Service endpoint
Interaction	Uniform interface	Service-specific interface
Semantics capabilities	Low (uniform interface)	High (arbitrarily complex)
Visibility	Through address and URL space description	Through service description
Awareness	Registry, search engine	Registry
Infrastructure	Simple	Complex

www.w3.org/) and OASIS (http://www.oasis-open.org/) make the SOAP suite, a complete set of standards for building SOA over the Web providing service description, cataloguing, security and so on. Indeed, this is currently the most spread solution for e-Business and e-Government systems.

SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment (W3C 2007). A SOAP Message is made of a Header and a Body. The optional SOAP Header element contains application specific information (like authentication, payment, etc.) about the SOAP message. The SOAP Body provides a mechanism for transmitting information to an ultimate SOAP receiver, that is the service provider. An important characteristic is that a SOAP Message could be transmitted using any protocol as long as it allows to transfer the serialized Infoset to the destination. Actually, several transport mechanisms (bindings) are defined for SOAP using application-level protocols such as HTTP and SMTP. This generality is obtained at the expense of the loss of protocol-specific characteristics. For example the HTTP binding utilizes HTTP as a transport-level protocol: the semantics of the request line and of most of the HTTP headers is actually lost.

Representational state transfer style

In the early 2000s W3C issued the document named ‘Web Services Architecture’ (W3C 2004a) which made SOAP the fundamental basis for a ‘new’ Web, a Web of exposed services instead of shared documental resources. But the great success of SOAP in many application fields like e-Business and e-Government did not guarantee the same success in other fields and applications characterized by different requirements. In fact, SOAP fits well to SOA, where different organizations expose complex services (e.g. banking transactions, travel reservations, commercial orders) implemented in background facilities which can be composed in workflows for carrying out high-level business processes. These great capabilities have the drawback of a complex infrastructure for services discovery, description, etc. But common Web applications, in particular the so-called Web 2.0 services, are light services dedicated to publish and access structured and semi-structured information (e.g. Web sites, Web interfaces to databases and repositories, Content and Document Management Systems, blogs, etc.), a context where SOA seems to be overloading. Basing on such considerations, the Web architecture underwent a deep reflection. In 2005, the W3C proposed a new vision of the Web architecture to make it conform to its original concept (W3C 2004b). Such architecture is based on an architectural style named REST proposed by Fielding (one of the designers of the original Web specifications) as the architectural style for the Web (Fielding 2000).

REST is a ROA style for distributed systems defined to describe the original Web architecture and to guide its future evolution preserving its fundamental characteristics – namely scalability. REST is defined starting from a set of constraints chosen to describe the way the Web works:

1. *Client–Server* interaction: in REST architectures there are two logical components with different functionalities: clients performing requests and servers which provide responses.

2. *Stateless* interaction: in REST architectures the server generates the response using only the information included in the request message. It cannot rely on server stored information; therefore, sessions are not supported.
3. *Cache* support: to improve performances, responses can be stored on end or intermediate systems for reuse.
4. *Uniform interface*: in REST architectures the client-server interface has four main characteristics:
   a. identification of resources, that is their addressability through proper identifiers;
   b. manipulation of resources through representations;
   c. self-descriptive messages;
   d. hypermedia as the engine of application state, that is the application is built following hyperlinks according to the navigation paradigm.
5. *Layered system*s: to generate a response a server can perform requests to other servers – acting as a client.
6. *Code-on-Demand* support: a resource representation can include code to be run on the client-side to improve capabilities – e.g. plugins for visualizing unsupported formats, client-side processing, etc.

Although REST is defined bearing in mind the Web, all the architectures satisfying the REST constraints are REST based architectures, not only the Web itself. For the same purpose of the term 'object-oriented', the term 'RESTful' was introduced (Richardson and Ruby 2007).

By our point of view, REStful architectures present two essential characteristics deriving from the Uniform Interface constraint:

1. All the significant resources are addressed and accessible through the same set of methods. This means that there are no resource-specific actions; possible actions should apply to all the exposed resources. Obviously this imposes to limit the possible actions to a small set of low-semantics actions which make sense for every resource in the information space. For example, they could be the four basic actions enabling the so-called CRUD pattern: *Create*, *Retrieve*, *Update*, *Delete*;
2. Logical connections between resources are made explicit as hyperlinks. A resource should be related to others through proper references using the resource identifiers. A REST application is realized moving along the hyperlinks to act on the target resources. This behavior maps the typical Web interaction based on the navigation (through hyperlinks) paradigm. What is transferred during the navigation is a representation (in a given format) of the resource state. This is actually the meaning of the name REST.

It is noteworthy that REST is not a technology. In particular, REST is not simply XML+HTTP which seems to be a common misunderstanding (Kelly 2007). In fact, a system could use XML, HTTP and other technologies in a REST way or not. What makes a system RESTful is not the adopted technologies but the way they are used, that is the architecture the system conforms to. Systems using traditional Web technologies (mainly HTTP and XML) in a non-REST way are usually referred to as POX-HTTP – Plain Old XML over HTTP.

Common RESTful architecture implementation

A Web system can implement a RESTful architecture in many different ways. Actually, only the adoption of URI addressing is mandatory. However, a common implementation is based on the use of HTTP as the unique application-level protocol. In this case, the HTTP verbs or methods: GET, PUT, POST, DELETE define the action to be performed on the target resource. These basic actions allow to implement the CRUD pattern: GET = retrieve; POST = create; PUT = update; DELETE = delete. In computer systems, the CRUD pattern has proven its validity when a low-level access to resources is required (e.g. file-system management actions, SQL for DB interaction). Nevertheless, other implementations are possible: e.g. ATOM (an XML-based Web content and metadata syndication format; http://www.ietf.org/html.charters/atompub-charter.html) defines a RESTful architecture implementing the CRUD pattern limiting the use of HTTP to GET and POST methods; Wikis (http://wiki.org) are basically RESTful systems.

RESTful architecture for earth sciences

A comparison between SOAP and REST architectures is just a subset of the general comparison between SOA and ROA. In general, it is not possible to say that one architecture is better than the other. The selection of the most effective system architecture depends on application requirements. Therefore, we need to have in mind a target usage case to analyse and discuss the effectiveness of the REST approach for Earth and Space Science (ESS). A meaningful usage case can be simply expressed as follows:

> An Earth and space scientist needs to share datasets in a simple way, for her/his usual research activities.

Typically, Earth and space researchers need to publish and access datasets in an easy way avoiding the trouble of maintaining complex technological infrastructures. Scientists know very well the application capabilities and logic they want to implement/use; while, they are less interested in the technology to build them. Common application capabilities include: to publish data and metadata (i.e. datasets), to discover and access datasets, to document them for processing, reference in research papers, and so on. For ESS applications, the conditions to set up and manage an information system is a major concern. An ESS research team is very different from a great enterprise or public administration needing to serve e-Commerce or e-Government applications. Typically, Earth and space scientists are not profound IT experts – actually, they are not required to be. Besides, they cannot often rely on full-time technical staff administrating a complex IT infrastructure. On the other side, they should stay focused on their research topics. One solution might be the outsourcing of data provision to advanced data centers; but this cannot solve the problem on the client-side. In addition, ESS researchers would like to utilize the same instruments they use in their day-to-day activities.

Consequently, the traditional Web approach is particularly appealing since it is well-known and is based on simple and extensible specifications which can be easily implemented in the existing modeling tools and applications. Anyway, to achieve user-friendliness, other important characteristics, such as scalability and overall performances, should not be sacrificed. This is the reason the REST approach seems to be particularly attractive for ESS applications. In fact, REST preserves the usability of the traditional Web, providing a solid architectural ground for future extensions.

RESTful implementation of coverage access services

In order to evaluate the feasibility of a RESTful architecture for ESS, it is useful to consider the existing geospatial services and evaluate which architectural approach they are based on and if and how they adopt or could adopt REST concepts.

Since data shared by Earth and space scientists are usually coverages (typically acquired from remote sensing systems), we will focus on two access services widely used by the ESS community: the OGC Web Coverage Service (WCS) and the NASA Community Data Access Protocol (DAP).

OGC WCS 1.0

The WCS defined by the OGC supports electronic interchange of geospatial data as ‘coverages’ – that is, digital geospatial information representing space-varying phenomena (OGC 2005).

The version 1.0 of the WCS specification will be considered, to avoid unneeded complexity in the discussion. Basing on this specification, we can extract a few logical entities representing the most important concepts for the coverage provision service. Figure 1 depicts a simplified information model of WCS 1.0 – expressed as UML Class Diagram. This shows five main logical entities:

- The *WCS Server:* it has some Capabilities and publishes an enumerable set of Coverage Offerings.
- The *Capabilities:* it expresses what the WCS Server can provide – i.e. the server properties.

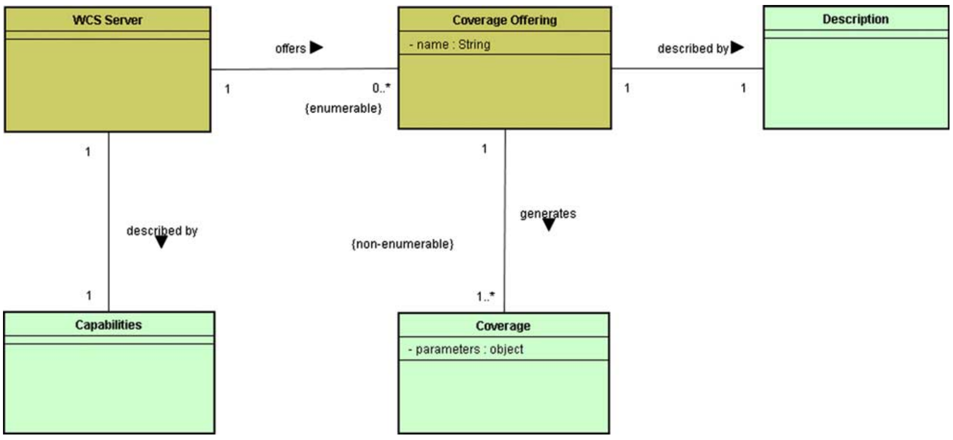

Figure 1. Simplified information model of OGC WCS 1.0.0. In dark color the hidden entities; in light color the exposed entities which are accessed through the WCS interface.

- The *Coverage Offering:* it is the logical generator of Coverages. A CoverageOffering can produce a non-enumerable set of Coverages by subsetting, interpolating, resampling, reference system transformations, etc.
- A *Description:* it expresses the properties of a CoverageOffering, especially describing the parameters which can be used to retrieve Coverages.
- A *Coverage:* this is the '*values or properties of a set of geographic locations*'. The WCS GetCoverage operation allows for the retrieval of Coverages from a Coverage Offering.

In Figure 1 the entities in dark color are not directly exposed by the WCS interface, while the entities in light color are accessed through the standard operations, namely: GetCapabilities, DescribeCoverage and GetCoverage.

In a SOA approach this information model is hidden in one or more computational elements which provide specific interfaces to it. The access is delegated to the computational elements which perform all the required interactions with the back-end information system in order to provide a response to the requestor. The following is the (partially implicit) approach as specified in the definition of WCS 1.0:

1. The WCS implementing component publishes a service endpoint exposing three operations – see the 'REQUEST' parameter;
2. Only the service endpoint needs to be addressed. This is the server URL.
3. No action can be performed on the internal resources other than the ones expressed by the interfaces;
4. The published interfaces are service-specific – they are meaningful only for an OGC coverage access service, not for accessing different resources (e.g. features, maps, documents) or for different types of coverage access services.

Figure 2 shows the WCS approach. Therefore, the interfaces do not need to be 'universally' known. Consequently, they must be described either in human-readable format (as for the KVP encoding in the specifications) or in a machine-readable format – as using WSDL for the SOAP encoding. It is remarkable that using the HTTP binding, either with GET method and KVP encoding, or with POST method and XML encoding, does not make the implementation RESTful. It is still a Service-Oriented approach, since the computational elements expose service-specific interfaces hiding meaningful resources. In particular, the semantics is contained in

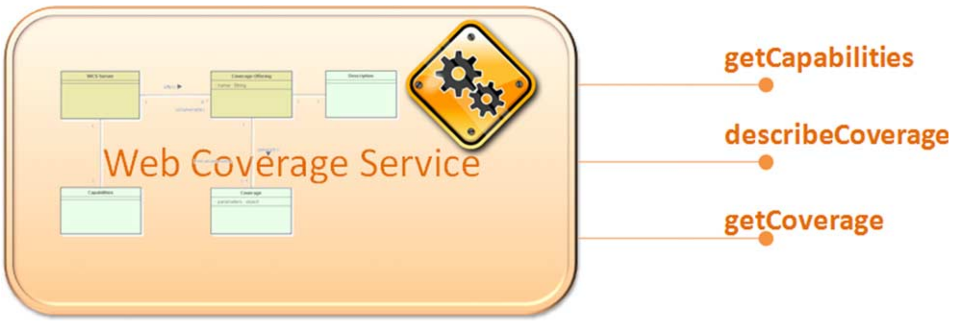

Figure 2. The service-oriented approach for implementing the OGC WCS.

the request parameters (message payload or in the query part) and not in the interface signature. Generally the OGC Web Services (OWS) specifications describe two possible implementations: SOAP-based and POX-HTTP. In comparison with the SOAP approach, the POX-HTTP approach provides some benefits such as the addressability of resources (with the KVP encoding) and the greater simplicity for service deployment and access. These two advantages are typical of RESTful systems, but since the underlying architecture is not REST-based, they are reduced by the lack of the other REST benefits. For example, since the interface is not uniform, a service description is required and a client should access the description before accessing the service, much like as in SOAP. Hence, in this service-oriented approach the system gains clear benefits from the full adoption of the SOAP suite; for example, the WSDL standard allows a complete service description, which provides the benefit of the availability of tools and libraries.

Figure 3 shows a possible hypothetical resource-oriented approach for implementing the WCS. Here all the significant resources (see Figure 1) are exposed through individual addressing and Uniform Interface. In Figure 3 the allowed

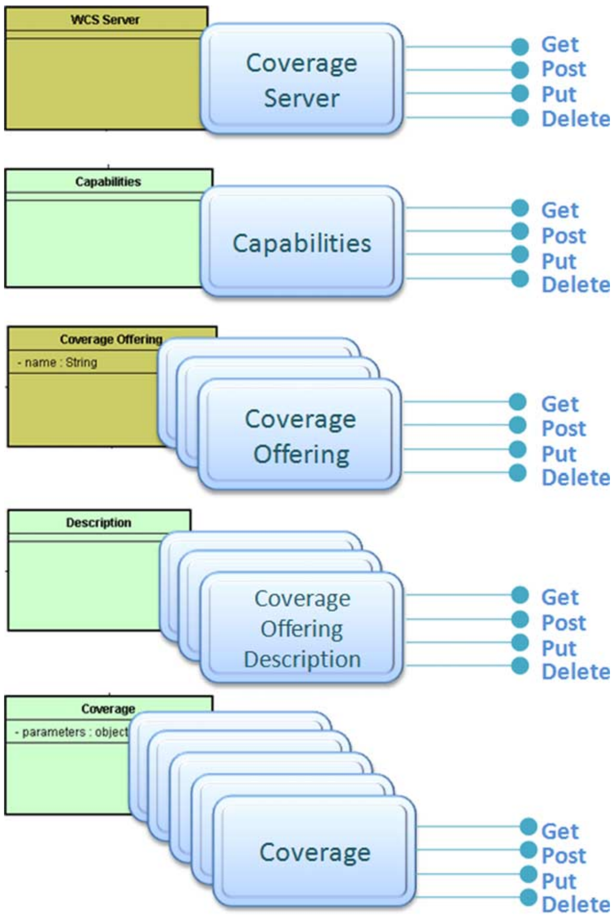

Figure 3. The possible resource-oriented approach for implementing the OGC WCS.

operations match the four HTTP verbs whose semantics conforms to the CRUD pattern. As each resource is accessed through the same methods, whose semantics is ‘universally’ clear, no operations description is required. The computational elements implement the basic application logic to realize each operation semantics for a specific kind of resources – e.g. what ‘update’ means for a Description element or what ‘delete’ means for a Coverage Offering element).

Identification and interaction

In WCS (and other access services) the difference between a service-oriented and a resource-oriented approach tends to blur, because the exposed services are intended to ‘access’, that is to retrieve, a resource. This is a basic service which can also be provided by a uniform interface based on the CRUD pattern. Therefore, the implementation of either a SOA or a ROA seems to reduce only to an interface rewriting. Nevertheless, much more problems arise when the systems need to be extended.

As far as the service and resource oriented implementation approaches are concerned, Table 2(a) and (b) try to summarize the differences in term of Identification (what is addressed) and Interaction (which actions are allowed). There exists a clear difference. The SOA approach defines a closed system with only one endpoint and a set of pre-defined actions. To extend the system capabilities, a definition of new operations is required. No limitation exists on the possible operations. In this approach, designers want that users access the resources only following a set of predefined ways, formalized by the exposed interfaces. This conforms to the guiding principles of SOA, which reflects the reality that ownership boundaries are a motivating consideration in the architecture and design of systems (OASIS 2006). In keeping with that, resources must not be exposed to the users. On the other hand, the ROA approach shows many more addressable entities which the user can interact with. However, the interaction is at a very basic level. More complex actions can be performed making use of workflow solutions. The user is allowed to build her/his own application and is responsible for implementing the corresponding application logic. It is remarkable that the system can be extended exposing other resources, without introducing new operations (unless they are considered meaningful for all the resources defined in the information space). Table 2(b) shows that even the four basic operations of the CRUD pattern could be easily used to implement effective actions that in service-oriented systems should be provided by specifically defined service interfaces. Valuable cases in point are: to update the description of a Coverage Offering element (i.e. PUT directed to a *Description* element) and to delete a Coverage Offering element (i.e. DELETE directed to a *CoverageOffering* element).

DAP 2.0

The DAP 2.0 is a NASA Community Standard for data transmission, designed specifically for science data. The concepts handled by DAP move around the Data Source entity:

- The *DAP Server* which serves Datasets from Data Sources.
- The *Data Source* which collects data organized as name-datatype-value tuples.

Table 2. – Different approaches for coverage access services. *Angle brackets ('<' and '>') enclose constant values; curly brackets ('{' and '}') enclose variables; square brackets ('[' and ']') enclose options (separated by* '|'*). QP stands for the Query Part containing parameters expressed as Key-Value-Pairs.*

(a)

Entities (Services)	URI	getCapabilities	describeCoverage	getCoverage
WCS Service	<end pointURI>	Retrieve capabilities	Retrieve description	Retrieve coverage

b)

Entities (Services)	URI	GET	POST	PUT	DELETE
WCS server	<baseURI>/ WCS	capabilities	–	–	–
Coverage offerings collection	<baseURI>/ WCS/cov- erages	contentMetadata capabilities section	*add an offering*	–	–
Coverage offering	<baseURI>/ WCS/cov- erages/{name}	offering representation	–	*update the offering*	*Delete the offering*
Coverage	<baseURI>/ WCS/cov- erages/ {name}?{QP}	coverage representation	–	–	–
Coverage offering description	<baseURI>/ WCS/cov- erages/ {name}/ description	description	–	*update the description*	–

- The *Dataset* extracted by the originating Data Source through constraint expressions (parameter selection, subsetting, etc.).
- The *Dataset Attribute Structure* (DAS) which characterizes the variables, their datatypes, names and attributes of a Data Source.
- The *Dataset Descriptor Structure* (DDS) which characterizes the variables, their datatypes, names and attributes of a Dataset.
- The *Data Dataset Descriptor Structure* (DataDDS) which holds data values along with the DDS.

In Figure 4, the entities in light color are accessed through the DAP standard requests. Although the diagram has some similarities with the one shown in Figure 1 for WCS, there are some important differences. In fact, DAP is explicitly based on HTTP and it uses the semantics of GET verb to express the action of retrieving resources representations. Although this conforms to a more RESTful architecture, some other characteristics are missing:

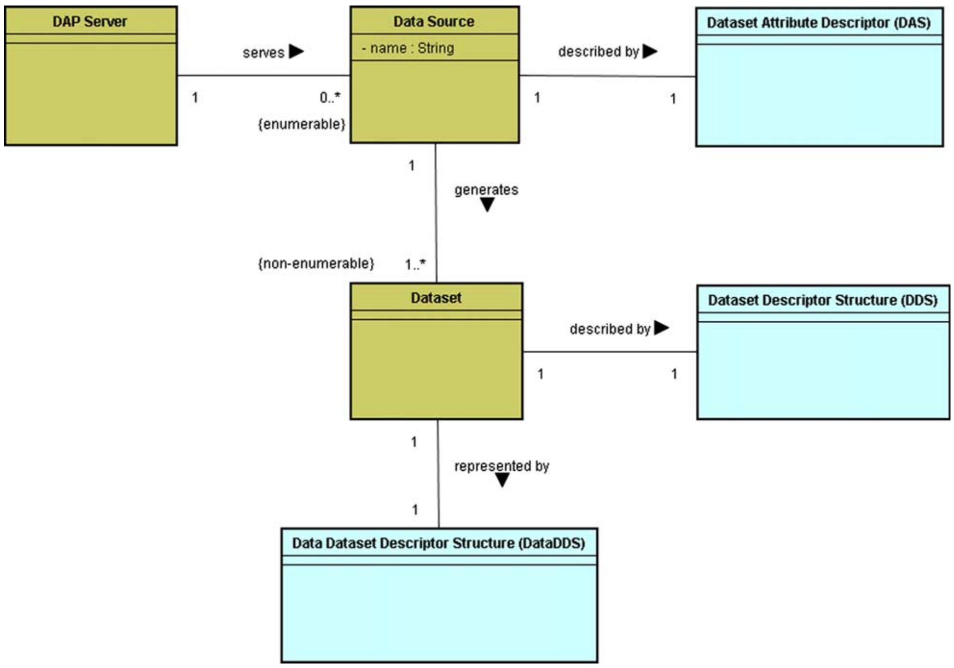

Figure 4. Simplified information model of an OPeNDAP implementation called DODS (Distributed Oceanographic Data System) 2.0. In dark color the hidden (implicit) entities; in light color the exposed entities which are accessed through the OPeNDAP interface.

- The generation of *Datasets* from *Data Sources* might actually hide a processing action. In addition to the basic constraint expressions (e.g. projection, selection, etc.) that generate *Datasets* simply filtering the originating *Data Source*, DAP specification states that: '*A constraint expression MAY also use functions executed by the server*' (NASA-ESDS-SPG 2007). OPeNDAP, a widespread implementation of DAP, relies on these 'Server Functions' for capabilities extension (OPeNDAP 2004a). Obviously, this choice implies a service-oriented approach introducing service discovery and interface description issues. Actually, the OPeNDAP developers are '*working out the details of the discovery and documentation mechanism*' (OPeNDAP 2004b).
- The main resources have not any hypermedia representations. Therefore, it is not possible to make explicit the relationships existing between the resources. However, some DAP implementations provide HTML representations. For example, OPeNDAP servers can provide an HTML *Data Source* representation including a form to generate *Datasets.*

In its basic form DAP is implicitly based on a resource-oriented approach. Thus, excluding the adoption of 'Server Functions' and with the introduction of hypermedia representations to support RESTful applications (and possibly supporting more basic operations such as the full CRUD pattern), DAP might conform to a RESTful architecture.

WCS and DAP were discussed in order to show how ROA and SOA concepts are actually present in the existing architectures and how a clear choice could provide benefits to both. However, a comparison between WCS and DAP is out of this paper scope. In fact, the architecture is just one of the important issues that may affect a successful choice. Actually, a main point to consider would be related to the semantics level mismatch existing between WCS and DAP services. WCS handles coverages, while DAP handles data. Hence, DAP works at a lower semantics level than WCS. This allows to claim the '*discipline neutrality of the DAP and the relationship between this and adoption of the DAP in disciplines other than the Earth sciences*' (NASA-ESDS-SPG 2007). Actually, the neutrality can be either a benefit (e.g. using the same service for accessing heterogeneous sources) and a drawback – e.g. no exact operation for specific data types is allowed, such as for specific geospatial data types.

RESTful implementation of geospatial web services

The ROA and SOA comparative analysis dealing with the coverage access services can be extended to most of the existing geospatial Web services. Besides, the introduced RESTful implementation recognized some general problems. For an effective RESTful implementation of Geospatial Web Services, a couple of important issues need to be carefully considered:

- Addressing: this includes:
  - Caching and the URL Space description;
  - RESTyling service-oriented specifications.
- Information Navigation: this comprises:
  - Hyperlink inclusion strategies.

Addressing: the URL space

Addressability, which is the capability of providing an individual address to every significant resource, is essential for web applications. Along with the uniform interface – that makes clear which actions can be performed, the addressability makes it easy for clients to use web sites in ways the original designers never imagined (Richardson and Ruby 2007). As previously outlined, addressing is the main difference between RESTful and service-oriented implementations. Because of the large amount of exposed resources, in a RESTful implementation, the definition of a simple addressing schema (URL Space) becomes important for usability and resource reference. For example, a URL like:

http://www.example.org/wcs;1.0.0/mycoverage?bbox=10.1,40.3,12.2,44.0&resx=150&resy=150

would be much more readable than the corresponding KVP encoded WCS request: *http://www.example.org/wcs?service=WCS&version=1.0.0&request=GetCoverage&format=GeoTIFF&resx=150&resy=150&bbox=10.1,40.3,12.2,44.0&coverage=mycoverage*

In this resource-oriented reference the required operation is not specified in the URL as in the service-oriented WCS POX-HTTP version (i.e. '*request=GetCoverage*'). Operations should be expressed in the method of the HTTP request. Moreover, some parameters (i.e. '*service=WCS&version=1.0.0*') are moved in the hierarchical part of the URL. Indeed, a RESTful implementation should make use of the full capabilities of Web specifications. Especially, URL has options which are often unused. For example, path parts parameters (e.g. the '*1.0.0*' after the semicolon in the example) can be used to specify options related only to specific parts of the URL – in this case '*WCS*'. Besides, taking into account the HTTP protocol specifications, format negotiation information (i.e. '*format = GeoTIFF*') should be included in the HTTP header (e.g. '*Accept*' field). Nevertheless, since Web clients usually does not allow users to modify request headers, it is a common practice to introduce specific query parameters to override HTTP headers (e.g. a '*mimetype'* parameter). In such a case a request like GET /resource?mimetype = image/png would be interpreted as GET /resource including the header field Accept: image/png. This also allows to include the format in the URI, which could be useful for citation purposes.

Caching and the URL space description

The URL HTTP Schema addresses an information space which is made up of a hierarchical part (including the server name, port and path) and a non-hierarchical part (including the query part after the question mark). The non-hierarchical part poses problems for caching, since query parameters can appear in a different order for different requests and a normalization is needed to associate URLs to the cached copies of documents. Thus, the use of hierarchical parameters, where possible, is really suggested. In general, this is possible when a resource is a collection of an enumerable set of other resources as for the Coverage Server and its Coverage Offerings. When the collection contains a non-enumerable set of resources they should be generally specified in the query part; an example is represented by a Coverage Offering and the Coverages generated from it. Figure 5 shows the hierarchical and non-hierarchical parts for the proposed WCS *RESTyling*. In the example, the WCS server has three Coverage Offering elements. Each element (i.e. resource) is characterized by an URL.

Due to the greater number of exposed resources, the URL Space of a RESTful system is much wider than for SOA systems which contain only service endpoints. Therefore, it should be described to users to make them aware of the exposed resources. Some specifications have been proposed, such as the Sun Web Application Description Language (WADL) to describe formally the URL Space handled by a server (Hadley 2006).

RESTyling service-oriented specifications

When systems follow a service-oriented approach (like WCS), *RESTyling* them is not simply a matter of rewriting their interface. For example, many WCS parameters are intended for guiding the interpolation and coordinate transformation internal processes; actually, they may be considered complex negotiation parameters and should be fully described to users. This makes these WCS parameters much like operation specifications. In a possible RESTful implementation, the Coverage Service element should provide coverages which are obtained by subsetting the coverage offerings and simply resampling them according to a pre-defined interpolation method and coordinate reference system. Different interpolation techniques and coordinate reference systems transformations depend more on users' needs rather than on provider's duties. Hence, they should be delegated to external and specialized services.

Information navigation: including hyperlinks

An underestimated yet important characteristic of RESTful systems is that the logical relationships between the resources should be made explicit through hyperlinks included in their representations – see Figure 5. In fact, the hyperlinks existence enables several capabilities and application scenarios; they include:

- The adoption of the navigation paradigm. As in the Web, a user could move from a resource to another one simply following the existing hyperlinks. Hence, a complete application scenario (search, view description, access dataset, etc.) could be implemented through hyperlink navigation.
- The information harvesting. A crawler application could follow hyperlinks to extract information about the available resources. This is the concept search engines are based on.

To implement these capabilities the resource representations should include hyperlinks. For example, a Coverage representation might include a reference to its generating Coverage Offering and to the Coverage Server, and so on. A problem arises with resources usually represented in binary formats (GALEON 2005). In fact, these formats might not accommodate hyperlinks – e.g. image formats like PNG; besides, they might be not a Web standard format; then, Web clients need specific plug-ins to interpret them – e.g. netCDF (http://www.unidata.ucar.edu/software/netcdf/), HDF (http://www.hdfgroup.org/) and so on.

Actually, it is not required that every resource representation includes hyperlinks, but that at least one of them does. Therefore, a possible strategy is to provide binary

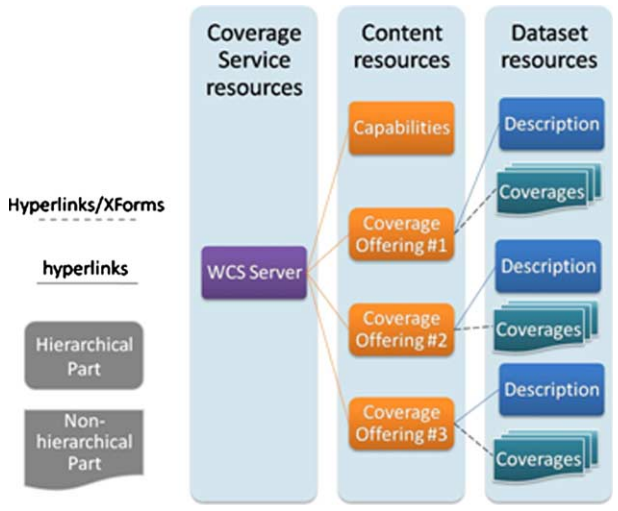

Figure 5. Hierarchical and non-hierarchical parts of the proposed WCS *RESTyling*. In this example the WCS server has three Coverage Offering elements.

representations along with a Web-safe alternative representation which includes all the needed references – for example an XML or HTML representation.

It is noteworthy that when references are numerous or even infinite it is not viable to provide all the hyperlinks. In such cases, hyperlinks can be built dynamically from user inputs using mobile code, e.g. JavaScript/ECMAScript (http://www.ecma-international.org/publications/standards/Ecma-262.htm) or simply forms, e.g. XForms (http://www.w3.org/MarkUp/Forms/). For example, a Coverage Offering representation could be an XHTML document including: (a) the Description of the Coverage Offering; (b) a hyperlink to the offering server (i.e. the WCS Server); and (c) an XForm accepting parameters to access the infinite set of Coverages generated from the offering – see Figure 5.

Towards a Geospatial Information Space: challenges and open issues

The adoption of a REST architecture for Geospatial Web Services would have an important benefit: the seamless integration with the Web. The traditional Web, but also the so-called Web 2.0, is inherently based on a resource-oriented approach. Therefore, the possible RESTful geospatial artifacts could be considered fully part of the entire Web. Geospatial resources can refer to other Web resources, such as documents and services, and vice versa. This approach would enable a Geospatial Information Space working much like the Web. This is particularly interesting for data curation and citation services. In fact, data publishing could be provided through a RESTful interface based on HTTP PUT method, ATOM Publishing Protocol or WEBDAV (Web-based Distributed Authoring and Versioning; http://www.ietf.org/rfc/rfc2518.txt) extension. While, data citation would be assured through the full addressability of resources in REST. Moreover, annotation services could be implemented using Semantic Web specifications (e.g. Resource Description Framework (RDF); http://www.w3.org/RDF/). Anyway, some issues need to be faced; they are briefly described in the next paragraphs.

Processing and sensor services

REST is a promising approach for implementing discovery and access services; in fact, they can benefit from traditional Web success developments, such as: document repositories and search engines. For other geospatial services, a RESTful implementation needs further analysis. This is especially true for the processing and sensor services.

Processing services

Processing capabilities are inherently service oriented. They specify complex tasks (e.g. transformations, simulations, modeling, etc.) to be performed accessing other resources. Therefore, their integration in a RESTful architecture raises semantic issues (what really a processing resource is?) as well as syntactic ones (how can parameters be expressed?). From a semantics point of view, a resource is a logical entity which could indeed represent a ‘processing service’. But the semantics to be applied to the uniform REST interface needs to be carefully defined, for a 'processing service' element. In other words: what is the retrieving of a 'processing service'? Is it a process representation or a representation of its output?

Concerning the syntax of a 'processing service', KVP parameters might be unable to express the complexity of service input parameters; therefore, an XML encoding should be used. Experiences from SOAP encodings should be the basis to address this point. Also experiences from REST interfaces for Cloud Computing systems should be useful (Chappell 2008).

Sensor services

Sensors are one of the main geospatial data sources, while the others are data repositories. Thus, the integration of sensors is fundamental to build an effective Geospatial Information Space. This requires to support both sensor planning and sensor observation services. Observation may require to access data in very different ways including streams, notifications and so on. While REST poses no limitations on the nature of resource representations, some technological issues may arise. In fact, the Web was mainly conceived for retrieving static or dynamic documents on user request. The main Web protocol, HTTP, is a request–response protocol which does not support streaming. Today, other Internet protocols are available for streaming services and could be adopted for sensors interaction – e.g. RTP/UDP (Real-time Transport Protocol; http://www.rfc-editor.org/rfc/rfc3047.txt). Besides, new services widely spread on the Internet show that the Web can provide access to effective streaming services – e.g. Web radio, video sharing, etc. However, an investigation is needed to carefully evaluate if they are able to satisfy the requirements of ESS Community.

Asynchronous services

In the ESS application domain some access services are inherently asynchronous. High-level services, such as updates notification, and alerts from ESS applications, might require an asynchronous infrastructure. Even basic access services could benefit from the introduction of an asynchronous infrastructure. A valuable example is the generation of a set of complex and huge coverages which is a time-consuming task. In fact, the output could be provided asynchronously to the requestor. In other cases, a geospatial dataset might be generated by a processing service which runs on a Computational Grid which implements an inherently asynchronous system. Therefore, an asynchronous infrastructure is a required capacity. HTTP, the Web main protocol, implements a synchronous pattern of communication. Anyway, several solutions have been designed for supporting asynchronous patterns. For example RSS, based on periodic polling of resources over HTTP, is widely adopted for simple notification services. Other solutions, like COMET, provide the right support to more complex asynchronous schemas (Crane and McCarthy 2008).

Machine-to-machine interaction

In keeping with the Web approach, the REST Uniform Interface constraints define the hyperlinks as the means for relating resources. This enables the adoption of the navigation paradigm to perform complex tasks and building applications. While this is a successful model for human-to-machine interaction – as demonstrated by the Web, it poses problems for machine-to-machine interaction. Actually, the flexibility of the navigation paradigm might be difficult to handle for machines. There exists the need to associate some semantics to hyperlinks and provide information about the target resources. This would be clear for humans but not for machines. Advances in the 'Semantic Web' (http://www.w3.org/2001/sw/) specifications are useful for this objective.

Caching

REST style is particularly suitable for read-mostly services, where cache can improve performances. Many geospatial resources (e.g. coverages) are often generated dynamically applying complex parameters; thus, it is difficult to associate parameters to a specific response. In fact, two requests could be syntactically different but semantically equivalent. They may differ for: (a) the order of request parameters; (b) parameter syntaxes – e.g. in a bounding box a latitude could be expressed as 2.0 or 2.00; and (c) the output formats specified by requests. Hence, common caching based on the association of a resource information and its generated response might be actually useless. In that case, the architectural basis remains unaltered but performances could be heavily affected. Smart caching techniques, based on parameters normalization, should be adopted. Besides, it could be useful to implement responses caching using intermediate format – and address format transformation issues.

Security

Another important issue is security. In order to support services and data policy, the provision of security services for access control, authentication, confidentiality, integrity, non-repudiation is required. While SOAP suite provides a complete and consistent set of specifications (i.e. WS-Security and related WS-* specification series), nothing similar exists in the traditional Web to be used in RESTful implementations. Once more, solutions can be found in IETF (http://www.ietf.org) and W3C specifications or in other existing standard extensions for the Web. For example, confidentiality and integrity can be provided at the transport-level by the SSL/TLS (http://www.treese.org/ietf-tls/) protocols, and at the message-level by multipart-encryption and multipart-signature (http://www.ietf.org/rfc/rfc2015.txt) standards. JA-SIG CAS (Central Authentication Service; http://www.ja-sig.org/products/cas) and OpenSSO (Open Web SSO project; https://opensso.dev.java.net) are examples of implementations of transparent Single Sign-On (SSO) authentication schemes.

Interoperation of service-oriented and RESTful geospatial services

As already discussed, the architectural choice depends on Community's usage cases and its critical requirements. Geospatial resources are needed for many applications domains – not only ESS. Thus, a RESTful implementation of geospatial services is not in competition with a service-oriented one. Traditionally, service-oriented implementations have been used for the integration of geospatial services in e-Government or e-Commerce infrastructures. In these cases, the adoption of SOAP provides a more complete and consistent set of specifications. However, where a POX-HTTP implementation is considered useful, a RESTful approach might be preferable.

Therefore both SOAP-based and RESTful implementations can coexist, as long as user requirements are made clear. Only after the definition of RESTful and service-oriented specifications for geospatial services, a reconciliation can be done at the technological level. For example, the same data repository might be accessed both directly, publishing a RESTful interface, and through delegation, deploying a SOAP interface.

Different approaches are possible to harmonize a service-oriented with a resource-oriented system. If resources exposed through a RESTful system must be accessed from a SOAP-based system, a specific component should implement a service-oriented interface carrying out, in the background, all the required operations on the resources. For example, this logical component could: (a) access a coverage; (b) apply an interpolation; (c) execute a coordinate transformation; and (d) provide the result to the requestor. Through the composition of basic actions on the published resources, the workflow could be arbitrarily complex and build a complete application – e.g. a model run. Adopting this approach, RESTful systems can be viewed as low-level systems used by high-level services to interact with the user; there exists a clear analogy with computer file-systems being used by computer applications.

On the other side, if a SOA service must be accessed through a RESTful system, some problems may arise. In fact, to simply provide access to the service in POX-HTTP, instead of SOAP, does not appear a good solution: as previously discussed, this implementation is service-oriented as well. A possible solution consists of providing a logical component which: (a) exposes a set of resources publishing a RESTful Uniform Interface; and (b) translates service requests into a set of Uniform Interface actions. Since services can be arbitrarily complex, the resulting RESTful interface might be not equipotent to the original service one. For example, to preserve the resource-oriented approach, coordinate transformation capability might be not supported; hence, responses are generated by calling the backend service and assuming a default coordinate reference system.

Conclusions

REST architectural style was conceived to describe the Web architecture and guide its future evolution. This aims to define, on a more stable ground, architectures characterized by the traditional Web scalability and simplicity. RESTful systems are open; in fact, providers directly expose resources leaving users to decide how to use them. This is very different from the SOA approach, where providers define exactly how to interact with the resources. In the ESS domain, and more generally in many other scientific research contexts, it is typically clear what must be shared – e.g the geospatial resources. On the contrary, there are few a-priory restrictions on how resources should be accessed and used. A flexible ESS resources sharing infrastructure should reflect this important aspect. Therefore, the REST approach seems very functional for typical ESS usage scenarios. The resource concept matches well the logical entities involved in most of the ESS usage cases – i.e. geospatial datasets, documents, etc., while no actions other than simple publishing and retrieving is pre-defined.

On the other hand, current international standards for geospatial data services are generally based on a service-oriented approach. In fact, this is particularly useful for different usage scenarios, including e-Business and e-Government ones. Therefore, it would be beneficial to define RESTful services beside the existing specifications. This task is not limited to a simple interface redefinition, as discussed in the present work. In fact, it includes a complete revision of what the service should provide. Moreover, while service-oriented approaches can benefit of the mature set of SOAP specifications, a RESTful architecture should be based on traditional and new Web technologies. In the recent years, many technologies have been introduced to address open issues and support new functionalities. However, several Web systems and services have been designed and developed without a clear architectural view. As a result, they often implement hybrid architectures using heterogeneous approaches and solutions. Hence, a deep investigation is required to verify the Web solutions which fit well in a REST-based geo-information system.

The adoption of the existing technologies would be a great benefit. In fact, avoiding to introduce new solutions, the hypothetical REST-ful geospatial information space would be integrated with the other existing Web systems gaining advantages from traditional and new Web services – like the so-called Web 2.0 services.

Notes on contributors

Paolo Mazzetti is a researcher at the Italian National Research Council (CNR). He is also contract professor of Telematics at the University of Florence in Prato. His main interest is in the Earth and Space Science Informatics with particular reference to the study and design of architectures for geospatial data sharing and processing. He is involved in several projects and initiatives for the design and development of Web and Grid infrastructures for Earth Science applications.

Stefano Nativi received the first and second (Laurea) degree in electronic engineering (telecommunications field) and the Ph.D. degree in "methods and technologies for environmental monitoring" from the University of Florence, Florence, Italy. He was appointed a PDRA Grant from the University of Bristol, Bristol, U.K. He is currently the Coordinator of the Earth and Space Science Informatics Laboratory, Institute of Methodologies for Environmental Analysis, Italian National Research Council, Florence, where he is also a Coordinator of the National Inter-university Consortium for Telecommunications Operational Unit. He is adjunct Professor of the University of Padua, Padua Italy, teaching "systems for land management" for the informatics specialis degree (Faculty of Mathematics). Dr. Nativi is the President of the Earth and Space Science Informatics division of the European Geosciences Union. He is a member of the IEEE Standards Working Group of ICEO and of IEEE GRSS Data Archiving and Distribution Technical Committee. He is co-PI of the OGC GALEON Interoperability Experiment (Geo-interface to Atmosphere, Land, Earth, Ocean netCDF) and a member of the Web Coverage Service standard working group. He co-leads the GEOSS IP3 initiative and is a member of the GEOSS SIF. He is a member of the "Metadata Core Drafting Team" for the Implementing Rules of the INSPIRE (INfrastructure for Spatial InfoRmation in Europe) European directive.

References

ANSI/IEEE, 2000. Std, ISO/IEC 42010 IEEE std 1471–2000, *System and software engineering-Recommended practice for architectural description of software-intensive systems.* Available from: http://ieeexplore.ieee.org/servlet/opac?punumber=4278470 [Accessed 24 March 2009].

Box, D., 2001. *A brief history of SOAP* [online]. Webservices.XML. Available from: http://webservices.xml.com/pub/a/ws/2001/04/04/soap.html [Accessed 4 February 2009].

Chappell, D., 2008. *A short introduction to cloud platforms: an enterprise-oriented view* [online]. Chappell & Associates. Available from: http://www.davidchappell.com/CloudPlatforms--Chappell.pdf [Accessed 4 February 2009].

Crane, D. and McCarthy, P., 2008. *Comet and reverse ajax: the next generation ajax 2.0.* Berkeley, CA: Apress.

Fielding, R.T., 2000. *Architectural styles and the design of network-based software architectures.* Thesis (PhD). University of California.

GALEON (Geo-interface to Atmosphere, Land, Environment, Ocean netCDF), 2005. *GALEON mailing list* [online]. Available from: http://www.unidata.ucar.edu/mailing_lists/archives/galeon/ [Accessed 4 February 2009].

Hadley, M.J., 2006. *Web application description language (WADL)* [online]. Available from: https://wadl.dev.java.net/#spec [Accessed 4 February 2009].

Kelly, B., 2007. *REST* [online]. UKOLN. Available from: http://standards-catalogue.ukoln.ac.uk/index/REST [Accessed 4 February 2009].

NASA-ESDS-SPG, 2007. *ESE-RFC-004.1.1. The data access protocol—DAP 2.0* [online]. Available from: http://www.esdswg.org/spg/rfc/ese-rfc-004 [Accessed 4 February 2009].

OASIS, 2006. *OASIS standard soa-rm. Reference model for service oriented architecture 1.0* [online]. Available from: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf [Accessed 4 February 2009].

OGC, 2005. OGC 05-076. *Web coverage service (WCS), Version 1.0.0 (Corrigendum).*

OPeNDAP, 2004a. *Open-source project for a network data access protocol* [online]. Available from: http://opendap.org [Accessed 26 January 2009].

OPeNDAP, 2004b. *An OPeNDAP quick start guide* [online]. Available from: http://docs.opendap.org/index.php/QuickStart [Accessed 26 January 2009].

O'Reilly, T., 2005. *What is web 2.0 – design patterns and business models for the next generation of software* [online]. Available from: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html [Accessed 4 February 2009].

Overdick, H., 2007. The resource-oriented architecture. *In*: IEEE Congress on Services, 9–13 July 2007. Salt Lake City, UT: IEEE, 340–347.

Richardson, L. and Ruby, S., 2007. *RESTful web services.* Sebastopol, CA: O'Reilly Media.

Thomas, E., 2005. *Service-oriented architecture (SOA): concepts, technology, and design.* Upper Saddle River, NJ: Prentice-Hall.

Vogel, W., 2003. Web services are not distributed objects. *IEEE Internet Computing*, 7 (6), 59–66.

W3C, 2004a. *W3C working group note. Web services architecture* [online]. Available from: http://www.w3.org/TR/ws-arch/ [Accessed 4 February 2009].

W3C, 2004b. *W3C recommendation. Architecture of the world wide web, volume one* [online]. Available from: http://www.w3.org/TR/webarch/ [Accessed 4 February 2009].

W3C, 2007. *W3C recommendation. SOAP version 1.2 part 1: messaging framework. 2nd ed* [online]. Available from: http://www.w3.org/TR/soap12-part1/ [Accessed 4 February 2009].