JRC Scientific and Technical Reports

Resource Oriented Architecture and REST

Assessment of impact and advantages on INSPIRE

Roberto Lucchi, Michel Millot
European Commission
Joint Research Centre
Institute for Environment and Sustainability

Christian Elfers
con terra - Gesellschaft für Angewandte Informationstechnologie mbH

EUR 23397 EN - 2008

The mission of the Institute for Environment and Sustainability is to provide scientific-technical support to the European Union's Policies for the protection and sustainable development of the European and global environment.

European Commission
Joint Research Centre
Institute for Environment and Sustainability

Contact information
Roberto Lucchi
European Commission Joint Research Centre
Institute for Environment and Sustainability
Spatial Data Infrastructures Unit
TP262, via Fermi 2749
I-21027 Ispra (VA)
ITALY
E-mail: Roberto.Lucchi@jrc.it
Tel.: +39 0332 78 5325
Fax: +39 0332 78 6325

Michel Millot
European Commission Joint Research Centre
Institute for Environment and Sustainability
Spatial Data Infrastructures Unit
TP262, via Fermi 2749
I-21027 Ispra (VA)
ITALY
E-mail: Michel.Millot@jrc.it
Tel.: +39 0332 78 6146
Fax: +39 0332 78 6325

Christian Elfers
Con terra - Gesellschaft für Angewandte Informationstechnologie mbH
E-mail: C.Elfers@conterra.de
Tel.: +49 0251 7474 333
Fax: +49 0251 7474 100

http://ies.jrc.ec.europa.eu/
http://www.jrc.ec.europa.eu/

JRC 45883

EUR 23397 EN
ISBN 978-92-79-09320-3
ISSN 1018-5593
DOI 10.2788/80035

Luxembourg: Office for Official Publications of the European Communities

Printed in Italy

Table of Contents

Abbreviations

HTTP	HyperText Transfer Protocol
ISO	International Standardization Organization
IR	Implementing Rule
OGC	Open Geospatial Consortium
ROA	Resource Oriented Architectures
SOA	Service Oriented Architectures
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
RPC	Remote Procedure Call
UML	Unified Modeling Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WFS	Web Feature Service
WPS	Web Processing Service
WSDL	Web Service Description Language

Introduction

In light of the emerging discussion on Resource Oriented Architectures (ROA) and REST technology platform as solutions for spatial data distributed infrastructures, the aim of this document is capture Resource Oriented Architectures principles and assess the feasibility as well as the advantages of using such approach compared with Service Oriented Architectures (SOA). Although most of the comments are domain independent we will use the INSPIRE [1] (and the OGC services [2]) infrastructure that is currently based on a SOA as domain for the comparative analysis.

The SOA and ROA architectural design patterns and the corresponding distributed programming paradigms provide a conceptual methodology and development tools for creating distributed architectures. Distributed architectures consist of components that clients as well as other components can access through the network via an interface and the interaction mechanisms the architecture defines; in the cases of ROA and SOA such distributed components will be named respectively resources and services.

In the following sections SOA and ROA principles are presented; considering that there is no common agreement on ROA principles the analysis takes into account different scenarios.

Service Oriented Architectures

A service provides business functions to its consumer and in ISO 19119 [3] it is defined as "*Distinct part of the functionality that is provided by an entity through interfaces*". In particular, in computing terms, a service is an application that provides information and/or functionality to other applications. Services are typically non-human-interactive applications that run on servers and interact with applications via an interface.

When designing a SOA the granularity and the various service types must be defined. Usually this phase is driven by the categorization of the business functions the infrastructure has to offer. In the case of INSPIRE (and OGC), service types (and Implementing Rules for INSPIRE) as well as the corresponding use case functionalities definitions are already available and can be reused.

Finally, although this section is devoted to discuss about the philosophy driving the design of SOA, some remarks about SOA and technological platforms are necessary. The way SOA is concretely developed depends on the specific platform and in particular on what concerns the communication infrastructure (usually based on TCP/IP such as HTTP) and on the way the interface is implemented and described. In the context of INSPIRE the platform is based on the SOAP (on HTTP POST) binding [4] while in the next section, when talking about ROA, we assume REST as development platform.

Resource Oriented Architectures

Main Concepts

Resource Oriented Architectures (ROA) [5] are based upon the concept of resource; each resource is a directly accessible distributed component that is handled through a standard, common interface making possible resources handling. RESTFul platforms [6] based on REST development technology enable the creation of ROA.
The main ROA concepts are the following (for a complete and exhaustive definition of the main concepts see [6]):

- Resource
  - anything that’s important enough to be referenced as a thing itself
- Resource name
  - unique identification of the resource
- Resource representation
  - useful information about the current state of a resource
- Resource link
  - link to another representation of the same or another resource
- Resource interface
  - uniform interface for accessing the resource and manipulating its state

The resource interface semantics is based on the one of HTTP operations. The following table summarizes the resource methods and how they could be implemented[1] using the HTTP protocol:

Resource method	***Description***	***HTTP operation***
createResource	Create a new resource (and the corresponding unique identifier)	PUT
getResourceRepresentation	Retrieve the representation of the resource	GET
deleteResource	Delete the resource (optionally including linked resources)	DELETE (referred resource only), POST (can be used if the delete is including linked resources)
modifyResource	Modify the resource	POST
getMetaInformation	Obtain meta information about the resource	HEAD

Table 1

In terms of platform implementation specification, each resource must be associated to a unique identifier that usually consists of the URL exhibiting the resource interface.

Designing Resource Oriented Architectures

One of the most important decisions that have to be taken in the design of a Resource Oriented Architecture is what must be considered a resource (by definition, each component deserving to be directly represented and accessed).

This is one of the main differences between ROA and SOA where in the latter one the single, directly accessible distributed component, represents one or more business functionalities that often process different potential resources. Such resources are credited candidates for being considered resources in a ROA, thus deserving to be represented as distributed components (e.g. features offered by WFS, registers or items of registries/catalogues, WFS and Registry/Catalogue services functionalities).

[1] Here we consider, as in the REST technology, the semantically equivalent ones.

Once having defined the granularity of the resources composing the ROA it is necessary to define, for each resource type, the content of the messages for invoking the methods as well as the corresponding responses. More in detail, beside the definition of resource types (and sub types), an addressing schema for accessing instances of those resource types, a response schema (response is not binary) and a mapping of logical functions to the HTTP operations. All resources in a ROA are accessed via the same common interface which is plain HTTP. It is worth noting that the usage of a common interface does not necessarily mean that all the necessary information enabling interoperability and collaboration among resources are available. The necessity of integrating a standard common interface description with some specific service instance aspects has been already addressed in other existing solutions such as, for instance, for the OGC WPS execute operation [7] where the specific processing detailed information are offered through the describeProcess operation. In a ROA, this information completes the description of the resources and their interface thus enabling: i) system integration and interoperability through tools for the creation of resource clients, ii) message validation processes, and iii) model driven development (top-down approach) for which a typed description of the interface and its operation is necessary.
In the REST technology such a description is based on WADL documents that play the same role of WSDL in the W3C Web Services platform; both languages use XML schema for expressing the structure of exchanged messages.

Technological considerations

Considering that REST and in particular ROA are at an earlier development stage this section is devoted to list some technological aspects that could be relevant for assessing the suitability of such solution.
At the best of our knowledge current development tools (e.g. JAX-RS, http://jcp.org/en/jsr/detail?id=311) are not addressing top-down development enabling model driven architecture development as well as resource client development. This approach was used in the ORCHESTRA project (http://www.eu-orchestra.org/) where UML service and data specification was used to derive the corresponding WSDL and related message schemas and then, through the WSDL, the client and service Java skeletons (apache axis tools).
However, the WSDL version 2.0 [8] now supports all the necessary HTTP operations (PUT, POST, GET, DELETE) thus making possible to describe the common REST interface, this means that WSDL could be used as the bridge between REST and W3C Web service platform enabling the inter-platform collaboration.
Finally a more practical consideration is about the HTTP GET operation that is supposed to provide access to resource representations. Although in principle there is not limitation to the length of the URL the browsers use a limited amount of characters, thus complex requests could be not easily expressed and used in normal web browsers and applications. Moreover, when the request message is complex the GET does not provide any mean for validating the structure as instead is possible when using, for instance, POST XML based requests (however, where used for getting information, would be in contrast with the ROA principles).

Implementing Resource Oriented Architectures

In this section we reason on costs and benefits of implementing a ROA starting from an already available SOA. Where necessary we will focus on OGC services and INSPIRE network services architecture that are both based on such distributed architecture model.
The debate on ROA (and REST) is spanning on different abstraction levels and in particular most of the discussion is focusing on REST rather than ROA design patterns. We present and reason on three different scenarios, two of them (REST) platform dependent and the other one aiming at reasoning at the abstract architecture level where ROA is conceived.

The cases we consider are the following:

- Weakly REST compliant architecture: it is based on distributed components offering their functionalities through the HTTP operations. The usage of HTTP operations could be semantically not coherent with Table 1.
- RESTFul compliant architecture: it is based on distributed components faithfully implementing the concept of common resource interface and of binding.
- Resource Oriented Architecture: it is composed by distributed components representing all the resource types deserving to be defined and exposed in the architecture.

It is clear that the first and the second category are principally related with REST and in particular with distributed component binding and interface aspects. These two views are in accordance with who is claiming that REST is a SOA development platform. The third one instead is more centring the architectural issue abstracting away from specific development platforms; this is the pure ROA where all the resources have an URI.

The analysis of the costs and benefits for each of the proposed scenarios, focusing where necessary on the existing INSPIRE SOA and on OGC services, is discussed in the following sub-sections.

Weakly REST compliant architecture

SOA based on services offering their functionalities through an interface accessible via HTTP operations are all members of this category. To exemplify, OGC Web Services specifications are all based on Remote Procedure Call programming style and the various service bindings (HTTP GET, POST, HTTP/POX, HTTP/KVP, SOAP via HTTP/POST, etc.) are all matching this definition. This is clearly a weak REST compliance level because it does not guarantee that a (semantically) common service interface is used but, instead, that the bindings are based on a subset of HTTP operations. Since this is exactly the reference architecture we use for the comparative cost and benefits analysis because it corresponds to what is currently available we will not reason further on this scenario.

RESTFul compliant architecture

It is worth noting that having a set of components using the same common interface does not necessarily produce a pure Resource Oriented Architecture. More precisely, we could obtain a RESTFul compliant architecture by an existing SOA simply by mapping services into resources (e.g. mapping the existing OGC services interfaces into the common resource interface) with no others resource types foreseen in the derived architecture. This is the case of the resource oriented view for OGC services discussed in [9] at the platform independent level (ISO General Feature Model). This view represents a sort of bridge between the Information and Service viewpoints and the resulting OGC platform independent specification can be used to derive the implementation specifications for both the W3C and the RESTFul Web services platforms.
The deviation between the derived architecture and the corresponding pure ROA consists exactly of the conceptual differences between SOA and ROA and it is well described in [6] where the authors say: "*Most SOAP services support multiple operations on diverse data, all mediated through POST on a single URI. This isn't resource oriented: it's RPC-style*". As previously mentioned the "RPC-style" is at the basis for most of the currently available SOA (even though some architectural alternatives to that service interaction pattern have been proposed) and it is in particular the case for OGC as well as INSPIRE services where the "procedures" correspond to the business functionalities the service is willing to offer. It is

worth noting that the fact that the cited text refers to SOAP services has no implications in this comparison, it holds for all SOA services based on RPC.
As far as development cost and benefits of such solution are concerned, we think that if on the one hand this solution does not require particular development efforts (it is mainly a matter of mapping service type interfaces into the common resource interface) and then the cost for transposing the existing INSPIRE (and OGC service) architecture is nearly irrelevant, on the other hand no clear benefits are in sight. The absence of evident benefits is particularly true for OGC services where, in order to properly interoperate and discover the right service instance, the statically available interface description has to be integrated with the getCapabilities document providing service instance specific meta information (and potentially it can be necessary use additional operation like getFeatureType for WFS or describeProcess for WPS): in this sense it is clear that supplying service functionalities through the common resource interface does not improve system interoperability and open-endedness.

Resource oriented architecture

In the context of INSPIRE and OGC architecture, resources do not necessarily include exclusively the services as conceived now. Potential meta types for the INSPIRE ROA are the ones defined in the INSPIRE Metadata Implementing Rules[2] [10]: service, dataset and dataset series. In [6] the following criteria is indicated: “when in Doubt, make it a Resource”. For instance, it is proposed to consider relations between resources as other resources; to exemplify this idea a datasets as well as a service able to use (visualize, transform or process) such datasets are meta types, and the “coupled resource” INSPIRE metadata element indicating the link between INSPIRE resources could be represented by an additional meta type. The proliferation of meta types compared with the (already well defined) service types causes a significant gap between the two architectures in terms of components and, therefore, most of the currently SOA-based specifications might not be re-usable for developing the equivalent INSPIRE ROA. Regarding the INSPIRE design and development process this aspect is particularly relevant, the development cost would be certainly bigger than adapting existing services and, in addition, the definition of the INSPIRE ROA might be time consuming and difficult to achieve with the planned time schedule. Moreover, the cost is not limited to the resource infrastructure development but is instead affected also by the necessary re-engineering of clients that are currently based on SOA. Regarding the benefits of such solution, the getResourceRepresentation offered through the HTTP GET makes the resource representation easily accessible and, consequently, general purpose search engines could more easily integrate resources representations into the search. Some proposals in this direction already exist like for the Dublin Core Metadata Element Set (http://dublincore.org/documents/2000/08/15/dcq-html/) where it has been defined a standard manner for introducing metadata in the HTML HEAD tag, this could be done in the RESTFul platform through the getMetaInformation resource method as well. However, unless the search engine is complying with that standard it would be impossible guarantee the usage of the exact metadata elements semantics (e.g. bounding box) and consequently support specific geo (and INSPIRE) search criteria. Finally, one of the claimed ROA advantages is the scalability that can potentially be better than SOA where services collect all the clients' needs[3].

[2] The fact that the Implementing Rules require, for each instance of such resources types, a Unique Resource Identifier makes reasonable assume that in a ROA these would become resources.
[3] This does not prevent the adoption of some solutions also in SOA such as the duplication of service instances and the usage of service brokers allowing balanced workload distribution among such services.

Applying ROA principles to INSPIRE: a simple case study

Based on this analysis it emerges that the only solution deserving to be further investigated is the one founded on ROA principles; as a simple case study we use the Draft Implementing Rules for Discovery and View services [11] for reasoning under the INSPIRE boundary conditions. However, given that this task requires a considerable effort and that no particular relevant strategic benefits have been identified we proceed with a preliminary analysis on the discovery service.

The resource oriented implementation offering the use cases supposed to be implemented with the discovery service could be based on a resource type, whose meta type is INSPIRE service, playing the discovery service role for the publish and discover use cases described in the IR. The records containing the metadata could be reasonably considered resources offering a way to get the representation (could also be more than one in case of different profiles), and to modify/delete the content. In this view the resource typed “discovery service” is a resource offering a mean for creating and searching resources whose type is “record”. More precisely, the publish use case can be implemented through the createResource method and it consists of the creation of a new record resource instance, with its own URL, containing the specified metadata. The discovery use case can be implemented using the getResourceRepresentation method of the “discovery service” resource type allowing for retrieving the list of URL addresses of all the matching resources from which will be then possible, in a second step, get the representation of each matching record by invoking the corresponding getResourceRepresentation method.
Let us now consider the required changes on the client side. We consider the case of a catalogue client using the getRecords operation offered by the OGC Catalogue service for supporting the discovery. In the resource oriented view, such a client should instead invoke the getResourceRepresentation method offered by the “discovery service” resource and then, for each matching record resource URL reference included in the response, it should invoke the corresponding getResourceRepresentation in order to get the record description (and it could be particularly costly in the case of long matching records lists). It is worth noting that the getRecords Catalogue operation provides three different result sets (brief, summary and full) that in the resource oriented scenario should be handled by the getResourceRepresentation method of record resources and not by the discovery service resource.
An alternative scenario, in order to reduce the responsibility at the client side, could be let the discovery service resource in charge of “invoking” the getResourceRepresentation for each matching record resource and return the result to the client exactly as in the getRecords operation. However, this would be again a service oriented interpretation where the service hides and centrally manages various resources while the resource oriented philosophy is having direct access to the source instead of to the black box handling the resource.
In general, centring the design on resources instead of on services requires more awareness and responsibility on the client side because the granularity (and, paradoxically, also the heterogeneity of resource types) of components with which interact is increasing. Moreover, when designing a ROA, the various functionality options that are normally aggregated at the service operation level in a SOA would be distributed over resource types as also emerged in the ROA based implementation equivalent at the getRecords result sets options.

Besides all these aspects, INSPIRE imposes some other requirements that we discuss in the following part. Finally, we conclude this section by reporting the state of the art about development tools.

Performance

In a distributed architecture, besides the computational cost of the offered functionality, the performance depends on the cost of transmitting data over the network (i.e. the amount of data moved between distributed components) and on the degree of scalability that the architecture has. These factors are depending on the communication infrastructure, that is the HTTP protocol, and on the solutions for scalability as, for instance, data and service replications, improved computational power of the servers and data compression. In general, as already pointed out, the fact that the ROA is characterized by a higher number of distributed components than the corresponding SOA ones is an advantage in terms of scalability. However, in the case of ROA it is worth pointing out that the cost of implementing the SOA service operations functionalities can be more complex and in particular it can imply more interactions over the network than the single service operation invocation affecting thus the single session performance as well as the system scalability. Let us consider again the ROA based implementation of the OGC Catalogue getRecords operation; as we illustrated in the client side implementation discussion such an operation requires, besides the invocation of the “discovery service” getRepresentation resource method, additional invocations at the matching record resources whose quantity depends on the number of matching records. It is trivial conclude that all these additional invocations can degrade the performance and the scalability even though they are performed (typically in parallel for the sake of performance) on resources distributed over the network.

Security

The most proposed solution for security is the usage of HTTP secure protocols (e.g. HTTPS; authorization and authentication mechanisms) that offer IP-to-IP secure solutions and not application-to-application ones. However, the fact that ROA and REST in particular are relying on HTTP is not preventing the usage of other protocols at application instead of at the communication infrastructure level. Authorization in ROA is typically done on the HTTP protocol level. By using the common HTTP interface and it’s GET, POST, PUT, DELETE operations, it is fairly easy to restrict e.g. DELETE-operations in general. But especially ROA implementations that are not strictly following the ROA principles are a potential security leak. E.g. if the GET operation with a query-string like “method=delete” is used instead of HTTP/DELETE.

Besides the canonical security issues, the peculiarity of ROA where new resources are dynamically created raises an additional problem that is defining who is responsible for assigning security policies to the created resources (the one willing to create the resource, the resource used to create the new one) that become themselves directly accessible over the network. This issue could be however coped by applying some secure policies based on addressing schema but probably this would not allow, for instance, determining the exact access rights of created resources using creator identity.

Conclusion

From the analysis it emerges that no particular benefits have been identified when using REST and RESTFul platforms for offering SOA services. Moreover, concerning pure ROA infrastructure we can certainly claim that it is not straightforward rephrase INSPIRE architecture (and OGC services) implement the equivalent ROA and it is important take into account the INSPIRE time schedule constraints. Moreover, real benefits have to be identified. Regarding interoperability between distributed components, as already stated the usage of a common standard interface in ROA does not exclude the need for more detailed contracts

properly describing the distributed components, i.e. the interface and the structure of exchanged messages which depends, if not on the specific resource instance, at least on the resource meta type. This makes necessary what in REST is often considered optional: REST resource must be necessarily described through the WADL (or an equivalent document) and the necessary message schemas. In this respect, there is not significant difference between REST and W3C Web Services description (WSDL) complexity. No significant difference also regarding the discovery.
Finally, an additional important aspect to take into account is the standardization process; if on the one hand standards related with SOAP are in advanced stage and the obligation for SOAP bindings to all new OGC service interfaces already exists (as decided June 2006, OGC 06-135r1, http://portal.opengeospatial.org/files/?artifact_id=17566), on the other hand the work on ROA and REST is still at an earlier stage (a OGC Service WG aiming to address related issues has been recently created).

References

1. European Parliament and European Council, *Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 Establishing an Infrastructure for Spatial Information in the European Community (INSPIRE)*. 2007.
2. OGC. *Open Geospatial Consortium Inc*. [cited 2006 May 17]; Available from: http://www.opengeospatial.org.
3. ISO/TC-211, *ISO 19119 Geographic information - Services*. 2005.
4. Network Services Drafting Team, *SOAP arguments (version 0.3 – 15.04.2008)*. 2008, European Commission.
5. Fielding, R.T., *Architectural Styles and the Design of Network-based Software Architectures, Ph.D. dissertation*, in *University of California, Irvine*. 2000.
6. Leonard Richardson, S.R., *RESTFul Web Services*. 2007: O'Reilly.
7. OGC, *Web Processing Service version 1.0*. 2008, Open Geospatial Consortium Inc.
8. W3C, *Web Services Description Language (WSDL) 2.0*. 2007, W3C.
9. OGC, *Integration of Resource-Oriented Architecture Concepts into OGC Reference Model (discussion paper)*, T. Uslaender, Editor. 2007, Open Geospatial Consortium Inc.
10. European Commission, *Draft Implementing Rules for Metadata (Version 3), Draft*. 2007, European Commission.
11. Network Services Drafting Team, *Draft Implementing Rules for Discovery and View services (IR1)*. 2007, European Commission.

European Commission

EUR 23397 EN – Joint Research Centre – Institute for Environment and Sustainability
Title: Resource Oriented Architecture and REST
Author(s): Roberto Lucchi, Michel Millot and Christian Elfers
Luxembourg: Office for Official Publications of the European Communities
2008– 16 pp. – 21 x 29.7 cm
EUR – Scientific and Technical Research series – ISSN 1018-5593
ISBN 978-92-79-09320-3
DOI 10.2788/80035

Abstract
In light of the emerging discussion on Resource Oriented Architectures (ROA) and REST technology platform as solutions for Spatial Data distributed Infrastructures, the aim of this document is capture Resource Oriented Architectures principles and assess the feasibility as well as the advantages of using such approach compared with Service Oriented Architectures (SOA). Although most of the comments are domain independent we will use the INSPIRE (and the OGC services) infrastructure that is currently based on a SOA as domain for the comparative analysis.

LB-NA-23397-EN-C

EUROPEAN COMMISSION

Publications Office
Publications.eu.int

ISBN 978-92-79-09320-3

9 789279 093203