ISPRS Int. J. Geo-Inf. **2014**, *3*, 1334-1351; doi:10.3390/ijgi3041334

OPEN ACCESS

ISPRS International Journal of

Geo-Information

ISSN 2220-9964

www.mdpi.com/journal/ijgi/

Article

The RichWPS Environment for Orchestration

Felix Bensmann *, Dorian Alcacer-Labrador, Dennis Ziegenhagen and Rainer Roosmann

Faculty of Engineering and Computer Science, University of Applied Sciences Osnabrück, Albrechtstraße 30, 49076 Osnabrück, Germany; E-Mails: d.alcacer@hs-osnabrueck.de (D.A.-L.); dennis.ziegenhagen@hs-osnabrueck.de (D.Z.); r.roosmann@hs-osnabrueck.de (R.R.)

* Author to whom correspondence should be addressed; E-Mail: f.bensmann@hs-osnabrueck.de; Tel.: +49-541-969-3186; Fax: +49-541-969-2991.

External Editor: Wolfgang Kainz

Received: 5 September 2014; in revised form: 20 October 2014 / Accepted: 24 November 2014 / Published: 5 December 2014

Abstract: Web service (WS) orchestration can be considered as a fundamental concept in service-oriented architectures (SOA), as well as in spatial data infrastructures (SDI). In recent years in SOA, advanced solutions were developed, such as realizing orchestrated web services on the basis of already existing more fine-granular web services by using standardized notations and existing orchestration engines. Even if the concepts can be mapped to the field of SDI, on a conceptual level the implementations target different goals. As a specialized form of a common web service, an Open Geospatial Consortium (OGC) web service (OWS) is optimized for a specific purpose. On the technological level, web services depend on standards like the Web Service Description Language (WSDL) or the Simple Object Access Protocol (SOAP). However OWS are different. Consequently, a new concept for OWS orchestration is needed that works on the interface provided by OWS. Such a concept is presented in this work. The major component is an orchestration engine integrated in a Web Processing Service (WPS) server that uses a domain specific language (DSL) for workflow description. The developed concept is the base for the realization of new functionality, such as workflow testing, and workflow optimization.

Keywords: geographic information system (GIS); spatial data infrastructure (SDI); web processing service (WPS); domain-specific language (DSL); orchestration

---

1. Introduction

Spatial Data Infrastructures (SDI) which partly evolved from service-oriented architectures (SOA) are becoming increasingly popular while staying conceptually compatible with the features of original SOAs [1]. The term SOA refers to a concept where functionality is provided by means of a set of reusable services. A key concept in SOA is the orchestration of web services (WS) that has been adopted for SDIs. The application of orchestration is especially useful in SDIs since calculations are typically long running and data intensive-and therefore more suitable for remote processing [2].

The Open Geospatial Consortium (OGC) [3] defines and maintains standards for geospatial web services that are dedicated to foster interoperability between geospatial services [4]. Therefore, the OGC relies on self-describing web services. The provided standards regulate what an OGC web service (OWS) has to look like, whereas the World Wide Web Consortium (W3C) web service standard Web Service Description Language (WSDL) only suggests how web services can be described. However the OGC standards provide support for web service standards like WSDL or Simple Object Access Protocol (SOAP), but they do not require their use. OWS operate directly on the Hypertext Transport Protocol (HTTP). Through application of the SOA principle they are intended to remain compatible with common web services. OWS offer operations just like common web services, although their structure, their exchanged messages and payload are predefined by protocol. Actually, the possibilities of an OWS exceed the possibilities WSDL offers [5]. All OWS have a common GetCapabilities operation that provides basic information about the service, the service provider and the provided contents [6]. However, the application of WSDL in SDIs is not continuously regulated by the OGC. In some cases OWS offer a WSDL description via the GetCapabilities operation; in such a case, parts of the provided information like the technical interface description are provided redundantly. Also to retrieve the description automatically a call has to be made yet making this concept obsolete. The type of an OWS defines its purpose and further interaction with the service.

Considering its description, the Web Processing Service (WPS) is a special case. WPS is intended for providing processing capabilities for geospatial data, but it can also serve as an interface to already available services for example. The service offers functionality by means of a process concept. A process contains server-side processing capabilities that can be discovered and accessed via the operations provided by the hosting WPS server. These operations are GetCapabilities that is inherited from common OWS, DescribeProcess, to retrieve detailed information on specific processes and Execute to start a process. SDIs are foremost dedicated to the provision of syntactic interoperability. WPS as the only OGC standard for general processing purposes is therefore crucial for any server-side calculation.

Using WSDL for binding to a WPS, not only the aforementioned OWS shortcomings apply but also the WPS specific extensions are not regarded, e.g., the service with its three operations can be described, just as its processes can. Here WPS faces a conceptual mismatch with WSDL. However, in common implementations this is solved differently. For example, in PyWPS [7] service and processes can be described separately, in 52° North WPS [8] one description is used solely for the service.

Orchestration in SOA is typically achieved by using web service orchestration engines (WSOE) in combination with a workflow description language; e.g., the Business Process Execution Language (BPEL). Often WSDL is involved as a supporting technology. Orchestrated services are offered by the orchestration engine. Though in individual cases this kind of orchestration could be applied to SDI,

problems are reported regularly [9]. However, tools and software that would support a convenient and applicable orchestration are still missing. Furthermore, workflows based on scientific data tend to require more complex workflow descriptions [10].

In general it can be stated that existing approaches of OWS orchestration depend on a high technology stack defined by W3C WS, BPEL and WSDL, with or without SOAP and trailing WS-* standards. Compatibility of OWS with existing W3C techniques is usually achieved by adding further adaption layers to the protocol stack, as happened with SOAP making the whole system more and more complex. While this enables interoperability with W3C WS, it decreases the conceptual coherence to other OWS and leaves advantages of OWS unused. Using BPEL for example may eventually result in the fact that current OWS orchestration approaches are not prepared to be applied by domain users due to complexity and the requirement for technical expertise in information technology.

The RichWPS research project aims at overcoming these shortcomings by simplifying the technical design as well as the user interaction. The key concept applied to this is a domain specific language (DSL) that closely integrates within the OGC provided technology stack, while unused degrees of freedom that common WS provide are closed out. Certainly, this results in the fact that IT/informatics experts will most likely experience an increasing distance to the new concepts and that a trend is followed, which diverges from the W3C WS tech stream. For compatibility reasons RichWPS will have to sort that out.

This paper presents the RichWPS orchestration environment and software collection. The prototypical environment is used for orchestration based on an orchestration engine in combination with a DSL. In the first step, the orchestration focuses on WPS. The orchestration engine is placed within a WPS server. It orchestrates WPS processes by chaining them and publishes the composition as WPS process, thereby making it available for further composition.

In addition to orchestration, RichWPS takes care of various further research issues concerning metadata history for processed data, workflow discovery, semantically aided search for workflow modeling, workflow debugging and the recomposition of existing workflows at runtime in order to achieve a better performance. Identifying the current shortcomings regarding overall user-experience, the developed software aims at providing a higher level of usability in order to push WPS forward into practical application. Further information about RichWPS can be found here: http://richwps.github.io.

Subsequent content describes the overall architecture focusing on discovery, modeling and the actual orchestration. The first chapter outlines related work and explains resulting design decisions for RichWPS. This is followed by the illustration of the most significant areas of work including workflow description and modeling concept. The adjacent chapter then concentrates on pending work by outlining the status of the project as well as future tasks. Finally a conclusion is drawn from the work in context of the research field.

2. Architectural Concept

As mentioned before, much research has been conducted in the area of orchestration. The conceptualization and implementation of geoprocessing workflows include different approaches ranging from transparent client side chaining of OWS [11] to the application of full-fledged opaque self-organizing workflow orchestration using semantics [2].

2.1. Mainstream Technologies

In the geospatial area the most common way to use OWS orchestration is WS chaining. Chaining, according to ISO 19119, is "A sequence of services where, for each adjacent pair of services, occurrence of the first action is necessary for the occurrence of the second action" [12].

Chaining can be applied using three different patterns: transparent, translucent and opaque chaining. The used patterns determine the amount of knowledge a user has of a certain service chain. At transparent chaining, the user is responsible for service discovery, evaluation and execution order of the chain; therefore he has access to all service details. At translucent chaining, the user uses a predefined service chain, offered by a WSOE, he is thereby aware of all participating services and may be able to poll status information from the participating services, though he has no influence on the execution. At opaque chaining, the chain is provided by a WSOE as a service and the user has no knowledge about the chain at all, he is just able to use it. Though chaining does not include control structures, parallel execution, alternative message transport methods, parameters in nodes or push or pull processing are possible [11,13]. So, the missing of control structures is what distinguishes chaining from orchestration in general.

When dealing with orchestration, especially in the form of translucent and opaque chaining, there are recurring issues that have to be taken into account. Particularly, when composing a workflow, descriptions of available services have to be present. When doing a manual modeling, there also needs to be appropriate modeling tools available, and finally a solution for workflow description and a way of providing it has to be available. Available systems solve these issues differently.

Many workflow architectures rely on the adaption of well-known and proofed techniques of SOA and the provided standards of the W3C and the Organization for the Advancement of Structured Information Standards (OASIS). Since some of these techniques are frequently applied to chain OWS, they are discussed here as well [13]. Next to the WS standard itself, the W3C first of all offers WSDL [14] to describe a WS's interface. Together with the WS*-standards, SOAP [15] adds to this to introduce more sophisticated features, e.g., for security [16]. In [17] the authors discuss and provide an overview on the latest development in the field of WS composition and WS technologies. Remarkable is, so stated, after 2006 research effort decreased. The authors of [18–22] treat earlier approaches towards these themes but exceed the scope of simple workflow modeling and execution by discussing comprehensive automated workflow composition based on semantics.

Several standardization bodies like Java specification Request 207, OASIS, Object Management Group and the Workflow Management Coalition invested effort into developing standards and workflow description languages. In [23] the author summarizes available WS orchestration technologies and their background. According to this, mostly orchestration is realized by some kind of workflow engine in combination with a matching workflow description language. The most popular workflow description languages are the XML Process Definition Language [24], the Yet Another Workflow Language [25] and BPEL. Further examples of languages are discussed in [26]. Because of its wide functionality, BPEL became the *de-facto* standard for (W3C-based) WS orchestration [17,27]. It is an XML based language that is executed by a BPEL engine like Apache ODE [28] and depends on WSDL. Since relying on XML BPEL is not intended to be used by humans, but there are graphical editors available [29].

Furthermore, there are script-based orchestration and choreography languages that are more suitable for human users and support advanced control structures. The most famous representatives are the Java

Orchestration Language Interpreter Engine (JOLIE) [30] and Orc [31]. JOLIE is a fully-fledged service-oriented programming language interpreter with according language. It is based on Java and offers a Java-/C-like syntax. By this means the language has computational as well as compositional aspects. It aims at offering a programming language for defining the base services, their organization in an SOA, and the behavior of the orchestrators responsible for the supervision of the interactions among the services thereby supporting various communication technologies, e.g., SOAP [32]. Scripts describing composed workflows can be offered as new WS by the interpreter engine. For developer support JOLIE comes along with a set of tools, e.g., for WSDL-to-code and code-to-WSDL or JOLIE-to-Java transformations. By supporting control structures it outgrows the requirements for chaining.

As another script-based solution, the Orc programming language aims at providing a language for task coordination with a manageable set of features; it therefore relies on an Orc-to-Java translator. The required WS interface definitions are extracted from WSDL documents whose URIs are specified in the Orc script. The resulting Java classes can be used in any Java application [33].

Finally there is the way of using plain Java or similar programming languages to program orchestrated workflows manually. Various tools are available that support users with WS stub generation. In the case of Java, this is the Java2WSDL-tool [34] and it is available in many IDEs. Besides this, it is also implicitly employed by the aforementioned Orc translator.

2.2. OWS Orchestration

Though OWS and W3C WS are both WS, they are not working the same way: while W3C WSs rely on SOAP RPC and WSDL, OWS use XML-RPC [35]. In consequence of that the OGC aims at providing better of interoperability [11,36]. So, common OWS offer in addition to their self-describing operations support for WSDL and SOAP what makes them available for W3C based orchestration approaches. Available approaches for translucent and opaque chaining of OWS focus on BPEL though their application in the OGC environment is prone [37]. Also, support for asynchronous calls cannot be granted [13]. Services that appear in OWS service chains are usually the Web Coverage Service (WCS), the Web Feature Service (WFS), the Web Map Service (WMS) as stated in [11] and additionally the WPS.

In [38] approaches to use various BPEL engines for chaining in context of a bomb threat scenario are described. The authors state that orchestration with the tested engines (ActiveBPEL 2.0, Oracle BPEL Process manager 10.1.2) is problematic due to several problems. Beyond other aspects those problems concern the provision of WSDL documents for OWS, the implementations of SOAP on OWS side and missing HTTP-binding support by the BPEL engines. The same is the case in [35]. In [39] these problems are stated as well and rehashed for overcoming by the OGC. In the provided examples the orchestrated workflows were offered to clients as W3C WS, but this may not be suitable in an SDI.

In the following papers the composed workflows are made available for use in an SDI by using an adapted WPS server that serves as an interface to the WSOE. It can, e.g. serve as a wrapper or as a proxy for a WSOE and publish the workflow as a process. By doing this, in an SDI, the workflow is accessible in an OGC compliant way and interoperability is not challenged. This method is described in [2,13,37]. In that solution a BPEL engine is approximated, just like described. Calls to the BPEL described workflow are directed at an adapted 52° North WPS server and redirected to the BPEL engine. Consequently, for applying WPS for workflow execution a mechanism for (un-) deployment is

implemented. The mechanism is called WPS Transactional (WPS-T) and it is considered for integration into the upcoming WPS 2.0 standard as an extension. By its initial conception WPS-T is able to deploy any type of resource including WPS processes.

By its wide definition a WPS implementation it is suitable to take over a WSOE's function in context of SDIs. Also, the standard was elaborated aiming at the support for chained workflows [5]. This way a workflow is available for reuse in another composition. Though using WPS is a practicable way to offer orchestrated workflows in SDIs, the issues stated above remain untouched.

Chaining can also be done by using the WPS specification solely, because it offers the possibility to nest requests successively into one another [5]. After having received the call, the first request is executed on server side. Then the remaining requests are forwarded to the next WPS instance, and so on. After all request executions have finished, the result is returned back along the call chain. This method is implemented in the Deegree WPS server [9] and in the Geoserver [40].

Yet another way of using pure WPS is the direct inclusion both of the call logic and of the clients into a WPS process at compile time [5,36]. The last two approaches have obvious shortcoming in flexibility, sharing options and the requirement of highly trained experts for the development of new WPS processes.

Another important field is the modeling of the workflows. As stated above graphical editors are available for most W3C-based orchestration languages. Full-fledged development environments for service composition like Taverna [41] have successfully been adapted to work with WSDL described WPS and feasibility was demonstrated [42]. This makes Taverna a considerable option for workflow modeling and a good example.

Finally, some issues in research that are also touched upon in this work are still eminent, like the inclusion of semantics in standard service description [13], the handling of large data transfers [27], service orchestration [43] and strategies to improve performance [13,43].

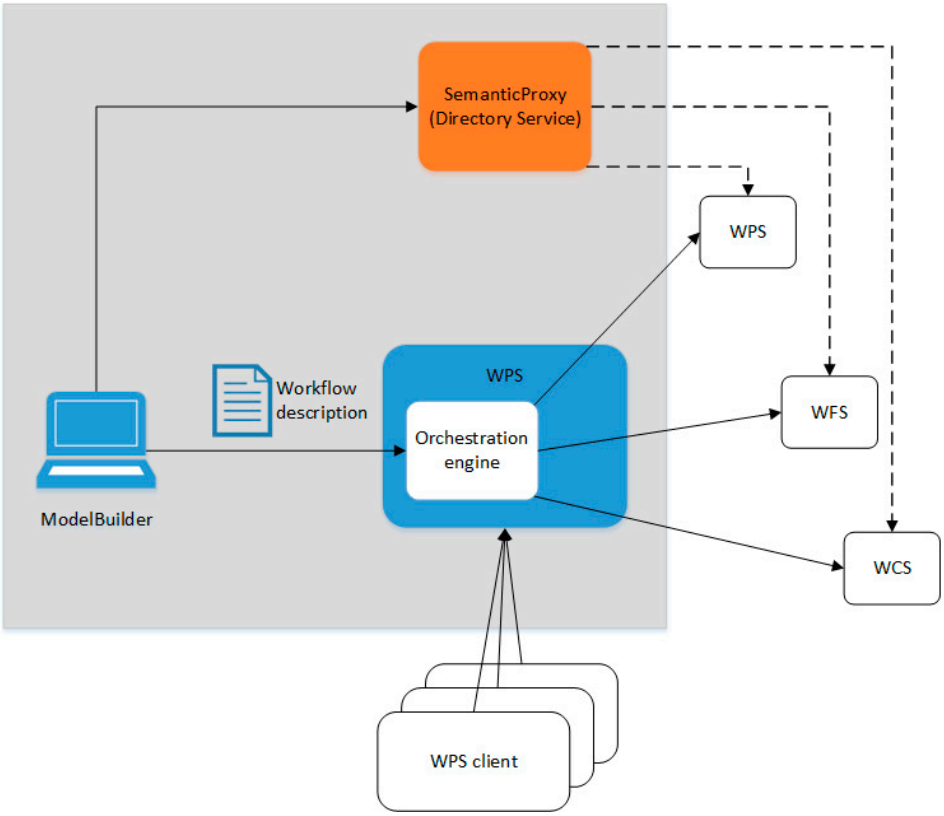
2.3. Architecture

The RichWPS orchestration environment takes up former approaches to implement an alternative way of WPS orchestration. It deals with three fields of work: discovery, modeling and orchestration. Based on these fields, three major components were developed to address these challenges. Figure 1 shows the components: the SemanticProxy, responsible for discovery; the ModelBuilder, responsible for modeling; and the RichWPS server, responsible for the actual orchestration. A dashed line indicates a "knows"-association while a drawn-through line indicate a "uses"-association.

Approaches resting on chaining by nested requests as given in the WPS specification were overruled, because resulting workflows are not available for other users or for use in further compositions. Also options for using control structures are missing.

Examined approaches that were using a general purpose language (Java, …) or a domain specific language (JOLIE, Orc) and manual implementation have been evaluated and in consideration of suitability for non-IT-experts discarded, though the provided functionality and support for control structures is promising. Another reason for this decision is the initially tackled use of WSDL for WPS description that would have to be solved here.

Figure 1. Overview of the RichWPS components.



Though classic solutions, e.g., based on BPEL are promising, it was found that the conceptual mismatches of W3C approaches and OGC approaches are potentially served better by developing a dedicated orchestration environment. Consequently, the main concept is to utilize a dedicated orchestration engine for the execution of custom workflow descriptions and use the provided properties of the OWS and WPS. Thereby the WSDL and SOAP issues are avoided as well as the minor incompatibilities with particular implementations. For deployment WPS-T is adopted from [37] and has been extended.

The used custom descriptions are optimized for using OWS. Therefore, a script based language, the RichWPS orchestration language (ROLA), was developed. ROLA is optimized for geospatial workflows and targets rather OWS orchestration and scientific workflows than W3C WS and business process workflows.

In the chosen strategy the development of a DSL is a key for overcoming the technology stack needed for classic orchestrations. The drawback of having to develop a new orchestration engine is accepted in this case. However, by integrating the orchestration engine directly into a WPS server that uses the WPS 1.0 interface and WPS-T for workflow deployment and undeployment, most of the expenses can be cushioned. Also, this solution fosters interoperability within the OGC-defined service environment as well as reducing complexity by avoiding the orchestration engine to be an additional component.

It may turn out that the ROLA is more suitable for use by domain users because it is designed to be close to natural languages in case the readability compared to XML-based definitions can be increased. Also, with the ModelBuilder assisting the user, modeling and deployment can be done without having direct contact to the ROLA. Therefore ROLA is compatible with its graphical variant. Taking the broad

nature of the interface into account, along with the free designable interior of the orchestration engine and the ROLA, RichWPS is open for more than chaining.

Besides, the presented approach provides a basis to address several of the recurring issues in context of WPS, e.g., for quality of service (QoS) improvements, a dynamic recomposition of workflows can be done, or processes located on the hosting WPS server can be accessed internally, without using the network communication. QoS thereby addresses a combination of qualities or properties of a service like availability or response time [44].

Nevertheless, to serve the orchestration engine, a modeling tool is required that supports the ROLA. Therefore, a graphical model editor called ModelBuilder is under development. The ModelBuilder acts as a client-side application to interact with the RichWPS server. Inspired by the Taverna Workbench [41], it can foremost be used for modeling workflows and to manage the lifecycle of the workflows. Furthermore, the ModelBuilder can be extended for testing, debugging, and performance analysis.

For proper functioning the ModelBuilder, as well as future services, require information about available services in the reachable network. It requires at least the interface descriptions of WPS processes for modeling. An extension of the process description allows the application of advanced searches. It can be assumed that comprehensive data collection at system start, e.g., by requesting preconfigured OWS addresses utilizing their self-descriptive abilities, will not prove to be a practical solution. The startup would be delayed and the user would have to know all available services. Also, the enrichment of the data with semantic information would not be usable in an efficient way. Therefore the data provision is outsourced into a directory service that is also accessible for other clients. The Resource Description Framework (RDF) is used to describe these services. RDF is extendable and allows a smooth transition into the application of semantically enabled operations like semantic searches. Applying the linked data principle, the service can also be integrated into existing RDF environments. This makes it incompatible with the Catalog Service for the Web (CSW) which would otherwise have been an option. The directory service component is called SemanticProxy.

Finally the RichWPS server and the SemanticProxy are conceptualized to be used in distributed environments like the cloud. Although, a thoroughly configuration is unavoidable especially with regard to the SemanticProxy and the linked data implementation, that requires that descriptions of RDF resources can be obtained from their identifying URI. With the deployment strategy chosen process deployments can be done by domain users solely and support by system administrators is only required at initial setup.

RichWPS forms a new approach of WPS that extends common WPS by new means of orchestration. An orchestrated WPS process is described and a procedure for deployment was determined. Since WPS firstly describes an interface, the standard is only affected by the WPS-T extension. Obviously it would be interesting to establish single features, e.g. ROLA or operations for testing and profiling in the WPS standard. Table 1 provides an overview of RichWPS in comparison to classical orchestration based on BPEL. It can be seen that most advantages that RichWPS shows lie in the interaction with OWS, whereas BPEL is strong in classic SOAs.

Table 1. Features of RichWPS compared to BPEL based solutions.

	RichWPS	**BPEL**
Interoperability with OWS	High	Low
Interoperability with W3C WS	Low	High
Provision of orchestrated services in SDI	High (WPS)	Low
Provision of orchestrated services in classic SOA	Medium (SOAP, WSDL supported but reported problems)	High
Service description	SemanticProxy, self-describing operations	WSDL from various sources
Interface of orchestrated services	XML-RPC, SOAP-RPC as inherited from hosting server	SOAP-RPC
Orchestration language	Customized DSL, with special regard for OWS	XML-based
Orchestration engine	RichWPS Server	Various
Tool support	ModelBuilder, for all tasks	Various, for different purposes
OWS Service discovery	High	Low
Usability	High	Medium
Control structures	Various (still open)	Basic set of operations available

3. Fields of Activity

With the concept at hand, the options for the implementation are presented in the following part. Significant areas of work are chosen to be presented in more detail.

3.1. Data Provision

Firstly, the data provision, meaning mainly the implementation of the SemanticProxy, is presented. The SemanticProxy offers a REST interface based on HTTP for requesting WPS description objects and process description objects, as well as for creation, update and deletion of objects. This way the data can be maintained. In later development iterations, an automated discovery is required in order to keep the service up to date. The micro web framework SPARK [45] is used for implementation.

The data exchange format is RDF/XML, one of the available RDF notations. For RDF related tasks, the SemanticProxy relies on OpenRDF Sesame [46]. RDF grounds on resources which can be described with statements that consist of a subject, predicate and object. Subjects, predicates and objects can be resources themselves, and therefore be described by other statements. That way the relation between resources in RDF can be represented as a graph. An object can also be a literal. A vocabulary is needed in order to have a base for statements. The vocabulary itself consists of statements. The W3C already offers vocabulary for certain domains. In further iterations, it is considered to employ existing vocabularies like OWL-S, but for now there is no scope for this. A vocabulary is created on whose base

the data is structured. Because RDF is not quite compatible with the concept of a directory service links to resources that belong to other descriptions are also thoroughly regulated.

A special property of RDF is that statements can refer to each other beyond system borders. This is possible because each resource is assigned a globally unique ID in form of a URI. The provision of information via this URI on a web server is a fundamental principle of linked data.

3.2. Modeling

The RichWPS ModelBuilder is the main software component for orchestration modeling. The orchestration of OWS is usually a rather complex task, which up to now, has to be done by users who have expert knowledge of computer science. To provide orchestration methods for other users, e.g., experts of geospatial domains, the ModelBuilder is designed to provide an easy-to-use GUI that simplifies the model creation and handling.

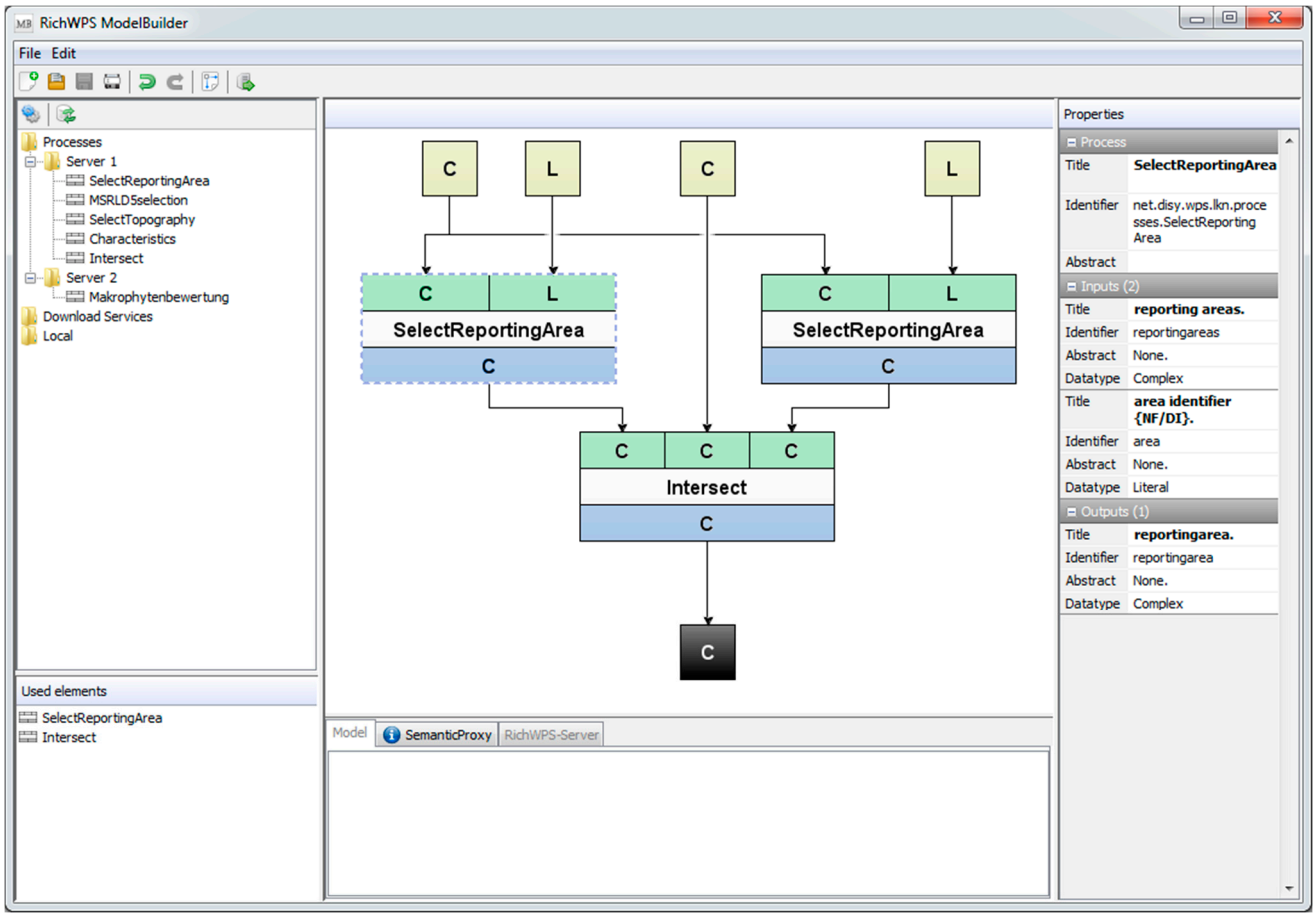
Initially it was examined whether the open source Taverna Workbench could be adapted to serve as the model builder. However, Taverna was designed for a wide area of application and needs large sets of software with functionality that is not needed by RichWPS. Also, relevant parts of Taverna, especially the GUI, would have too many dependencies to be reused. So it was decided to develop a custom small and more agile application.

With the ModelBuilder, the orchestration procedure is abstracted and can be designed in a graphical model editor, which is the main GUI component. It enables users to graphically arrange elements which represent existing download- and WPS-services. By connecting the elements, the desired work- and dataflow can be designed. The notation used for the diagram is transferable into the ROLA. A screenshot with an example model is displayed in Figure 2. A process is represented by a box with its title drawn on it and its inputs on the top and its outputs on the bottom. The letters on the inputs and outputs indicate whether a port uses a literal, a bounding box or a complex data type. Inputs and outputs of the surrounding workflow are presented as square boxes usually at the top and the bottom of diagram.

Services may require multiple input parameters and may provide multiple data outputs for the result, so users are able to connect specific process in- and outputs. Common editing actions, like adding, moving and deleting elements, are offered as well. Usability aspects are further improved by an integrated undo/redo mechanism.

The ModelBuilder can be configured to use semantically enriched process and service information from the RichWPS SemanticProxy. This information describes functional and non-functional properties of existing services and processes, which enables users to search for and select appropriate WPS processes during the modeling phase. A tree view displays available WPS processes. Users can easily add them to the current model by double-clicking the tree view node or by dragging-and-dropping nodes to the model editor.

Figure 2. Screenshot of the ModelBuilder with a graphical workflow model.



The modeling procedure is supported by displaying context-sensitive information. Amongst others, this can be process information (identifier, abstract, *etc.*), but also hints for indicating the validity of connections. The visual aids are accompanied by logging windows, which allow the user to retrace and follow application events. Additionally, tooltips are used to provide specific detailed information for GUI elements without overloading or messing up the user interface. Created orchestration models can be saved to and loaded from the file system.

A validation mechanism is included in the ModelBuilder to check the data type matching and dependency structure. Once validated, orchestration models can be saved, updated and deployed on the RichWPS server. To achieve this, the model data is automatically translated into the ROLA. Therefore the dependencies between the processes are analyzed and an execution order is determined. The serialized form can then be translated into ROLA step by step. In case of two or more parallel executable processes an arbitrary order between these processes is determined. Then on the server side, the orchestration engine interprets the ROLA script and realizes a new (or updated) WPS process by using and executing the arranged, existing WPS processes. A WPS client application can be used to execute the created process. Also, the execution can be invoked within the ModelBuilder. The ModelBuilder is furthermore client-destined for testing/debugging, profiling and process lifecycle management.

3.3. Model Description and Execution

The interface between the RichWPS ModelBuilder and the RichWPS server is based on WPS-T [37]. It is used in an adapted form of the upcoming WPS 2.0 extension WPS Transactional (WPS-T, unpublished work/draft). It is of significant importance because it is the precondition for applying the moving code paradigm [47]. It not only enables simple process deployment, but also (semi-) automatic deployment and thereby can realize dynamic environments to improve performance (e.g., bring the code to the data) and share functionality by realizing ad-hoc services. The extension itself provides additional operations for on-the-fly deployment and undeployment of processes. Therefore, the mandatory operations DeployProcess and UndeployProcess are introduced. Support for different types of process code is achieved by using so called DeploymentProfiles and according DeploymentSchemas, which are accessible whilst discovery. The deployment of a process takes place by providing the two portions: (1) The ProcessDescription and (2) the ExecutionUnit. The ProcessDescription is used to define and contract the process interface by means of the process identifier and the necessary in- and outputs, as defined in the WPS specification. The ExecutionUnit can be any type of data, e.g., an executable binary or script-file. Whether the server supports a format or not is defined by the available DeploymentProfiles.

In context of the RichWPS project, the description of the ExecutionUnit is realized by using the ROLA and an according DeploymentProfile. ROLA is designed to be process- and data-centric and to omit redundancies that could occur when mixing the domains of (1) interface contraction and (2) workflow description. The realization is based on an external DSL rather than on XML or given workflow description languages.

A textual workflow representation was realized by using an external DSL, which, compared to an internal DSL, allows to produce even more streamlined language [48]. That language is based on a custom grammar, which can be adopted for different software environments. However, the choice of an external DSL over an internal DSL, XML or given languages, directly impacts the upfront design and implementation effort.

To maintain readability, the workflow representation is supposed to be a mere sequence of calls to available processes. Though, considering the later interpretation, the textual representation contains the necessary information for, e.g., parallel processing. As described in the chapter above, the according graphical notation enables the visual modeling. Compared to the intended textual representation, the graphical notation contains a higher degree of information; a circumstance that occurs by splitting that information into the above named domains and artifacts (1) & (2).

Overall, the implementation of the language takes place in an iterative procedure. The first iteration focuses on the basic language elements, necessary for achieving simple process chains. The second iteration addresses the parameterization and more extensive use of the WPS protocol in order to implement more complex process chains, and to take advantage of the given specification. However, in order to approach a more scientific modeling, certain aspects such as adaptiveness, conditions, loops, basic arithmetic and exception handling need to be considered as well [10]. Also, when thinking of data-sources, the fine-tuned selection of specific datasets, e.g. selecting the n-th element of a collection, have to be considered. Therefore, the third iteration focuses on more advanced concepts of workflow modeling.

3.4. Language Concept and Implementation

The language design rests on three key concepts: (1) Locally available and remotely bound processes need to be explicitly distinguished in order to be able to enhance execution; (2) Bindings for remote and local processes can potentially be exchanged by equivalent processes; (3) Interface contraction and the workflow description are considered to be two individual domains, even though they share common elements and interdepend each other. Separating those concerns aims at maintaining readability of the script, and is used to hide protocol complexity wherever possible.

For now, all those measures lead to lean textual manifestation that deals with process chaining by providing four major language elements. (1) References are used to refer to runtime elements, among these are in- and output parameters defined by the interface description, and free-to-choose variables that store intermediate results and values.

(2) Assignments are used to set output parameters or variables. The use of assignments is subject to restrictions, e.g., values of input parameters cannot be altered at runtime. In combination with references and most basic datatypes, assignments are used to steer the data flow right up to output parameters; (3) Bindings are used to declare processes, and to define their respective endpoint. They are identified and referred to by unique shorthands. Bindings are distinguished in local and remote bindings. By using local bindings, the runtime environment responsible for execution is instructed not to use exterior interfaces for process calling. Remote bindings are furthermore defined through an endpoint which is identified by using the classical URI.

Finally, (4) Execute Statements are used to bring together references and bindings in order to execute WPS processes. The execute statement itself is divided into two sections for input and output parameterization. Unlike assignments, these sections are used to connect given runtime references with process specific parameters.

Table 2 shows an overview of the language elements.

Table 2. Overview of language elements.

(1) References	(2) Assignments	(3) Bindings	(4) Execute Statements
in.wps-input-identifier	**out**.wps-output-identifier = **var**.unique-identifier	**bind process** wps-compliant-identifier **to** org/shorthand	**execute** org/shorthand **with** **in**.example **as** INPUT1
out.wps-output-identifier	**var**.unique-identifier = **in**.wps-input-identifier	**bind process** http/https, port, path, wps-compliant-identifier **to** org/shorthand	**var**.example **as** INPUT2 **store** RESULT1 **as out**.result1 RESULT2 **as** **var**.variable
var.unique-identifier	**var**.unique-identifier = “value”		

As with this publication of this article, an early prototype for evaluating the concept has been realized. Its implementation is based on Java as programming language and the Xtext [49] framework for generating parsers for ROLA-scripts. Parsing and execution takes place at the orchestration engine which is embedded within the 52° North WPS Server V. 3.3.0.

3.5. Monitoring and Dynamic Reconfiguration

Due to a frequently expressed need for QoS enhancement of WPS processes and geospatial workflows, a monitor for WPS processes is currently under development. Scheduled execute requests are done to WPS processes, and response time and availability is recorded and published via the SemanticProxy. Other components may read these data and compare it to their needs in order to determine readiness of use. Major problems to deal with are the question where to get the test data from and how to validate the correctness of processing results.

A component that can use the performance data for service enhancement is the ModelBuilder with which the user can use the data to determine whether a process he considers for modeling meets the requirements or not. Another one is the RichWPS server. Assuming that the server knows processes that are equivalent to processes in its workflow descriptions, it can substitute processes in the workflow automatically with processes that provide a better performance. Precondition for this is a semantic match-making to find functional twins. This could happen on the SemanticProxy.

4. Outlook

Currently RichWPS provides a software collection and orchestration environment for OWS focusing especially on the WPS. The three major components SemanticProxy, ModelBuilder and RichWPS-Server interact with each other to realize a user friendly environment. A workflow description language was introduced that aims at reducing technology and usability complexity by assuming a geospatial context. While basic operations like discovering, modeling, deployment and workflow execution have been realized prototypically some work is still open.

The next step following in the RichWPS project will be the extension of support for further OWS especially WFS and WCS, as for the moment the focus is on WPS.

Furthermore, more effort will be invested in the automatic reconfiguration of orchestrated WPS. As briefly explained in the previous chapter, the optimization should happen at runtime with respect to QoS data that will be provided by the monitor. This includes also strategies for the realization of the semantic match-making.

Debugging is a new field in the considered area. It can be applied on orchestration level, in that interim results after the execution of single processes can be made available whilst further execution of the workflow continues. Applying detailed execution controls like breakpoints or step-by-step execution is not intended. A single call collecting interim results and returning them after workflow execution is a pragmatic first step. This issue will be treated later.

In the same way, performance measuring of the execution times can be done as part of the testing. Profiling a workflow returns results that identify poor performing processes way more comfortably and faster than a monitor would do while also reducing configuration overhead for single processes.

Cooperation partners emphasize that there is a need for extensions for the documentation of processing results with metadata. Because the extent of the documentation and applicable concepts is currently not elaborated, this aspect is too extensive and needs to be addressed in a further project.

5. Conclusions

OWS orchestration using a WSOE in combination with, e.g., BPEL has significant drawbacks. The RichWPS orchestration environment realizes a new approach of OWS orchestration. By addressing especially OWS, RichWPS overcomes the challenges of workflow modeling by using a DSL that takes advantage of the OGC provided specifications to simplify the workflow design significantly.

RichWPS provides the necessary software components and supports the user in workflow modeling tasks. Information about available OWS is necessary for workflow modeling. In RichWPS it is provided from a directory service that also offers options for semantic enabled search. This directory service is called SemanticProxy. Workflow modeling can be done with the ModelBuilder. It provides a sophisticated graphical diagramming language that is optimized for geospatial workflows. The actual orchestration is realized with the RichWPS server. Once the modeling is completed the ModelBuilder can be used to deploy and undeploy workflow models onto the RichWPS server. Therefore the graphical model is translated into a textual DSL. Once deployed the RichWPS server makes the workflow available as a WPS process.

With the DSL and the ModelBuilder the requirements for simplicity and usability are fulfilled. Also fulfilled are adaptability and interoperability since the overall concept offers several options for new features like debugging or performance enhancement of workflows. At the same time the main component, the orchestration engine remains fully compatible with the OGC defined policies for geospatial web services by offering WPS interfaces for interaction.

Acknowledgments

This work has been funded by the German Ministry of Education and Research (BMBF) (RichWPS).

Author Contributions

This paper is a combined effort by Felix Bensmann, Dorian Alcacer-Labrador, Dennis Ziegenhagen and Rainer Roosmann. The author order reflects the extent of individual contributions.

Conflicts of Interest

The authors declare no conflict of interest.

References and Notes

1. Müller, M.; Bernard, L.; Brauner, J. Moving code in spatial data infrastructures-web service based deployment of geoprocessing algorithms. *Trans. GIS* **2010**, *14*, 101–118.
2. Foerster, T.; Schäffer, B.; Baranski, B.; Brauner, J. Geospatial web services for distributed processing—Applications and scenarios. In *Geospatial Web Services: Advances in Information Interoperability*; Zhao, P., Di, L., Eds.; IGI Global: Hershey PA, USA, 2011; pp. 245–286.
3. OGC. OGC Main Page. Available online: http://www.opengeospatial.org/ (accessed on 17 October 2014).

4. OGC. OGC Standards. Available online: http://www.opengeospatial.org/standards/is (accessed on 17 October 2014).
5. Schut, P. *Web Processing Service V 1.0.0*; Open Geospatial Consortium: Wayland, MA, USA, 2007.
6. Whiteside, A. *OGC Web Services Common Specification*; OGC Document; Open Geospatial Consortium: Wayland, MA, USA, 2007.
7. Cepicky, J.; Becchi, L.; Casagrande, L.; Holler, S.; Skintzos, P.; de Jesus, J. PyWPS. Available online: http://pywps.wald.intevation.org/ (accessed on 5 August 2014).
8. 52°North Initiative for Geospatial Open Source Software GmbH 52° North WPS. Available online: http://52north.org/communities/geoprocessing/wps/ (accessed on 5 August 2014).
9. Stollberg, B.; Zipf, A. OGC web processing service interface for web service orchestration. In Proceedings of the Web and Wireless Geographical Information Systems 7th International Symposium, W2GIS 2007, Cardiff, UK, 28–29 November 2007.
10. Akram, A.; Meredith, D.; Allan, R. Evaluation of BPEL to scientific workflows. In Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), Singapore, Singapore, 16–19 May 2006; Volume 1, pp. 269–274.
11. Alameh, N. Chaining geographic information web services. *IEEE Internet Comput.* **2003**, *7*, 22–29.
12. *ISO/TC ISO 19119:2005—Geographic Information—Services*; International Organization for Standardization: Geneva, Switzerland, 2005.
13. Schaeffer, B. *OGC OWS-6 Geoprocessing Workflow Architecture Engineering Report*; Open Geospatial Consortium: Wayland, MA, USA, 2009.
14. W3C Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. Available online: http://www.w3.org/TR/wsdl20/ (accessed on 17 October 2014).
15. W3C SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Available online: http://www.w3.org/TR/soap12-part1/ (accessed on 17 October 2014).
16. *OASIS Web Services Security : SOAP Message Security 1*; OASIS: Burlington, MA, USA, 2006.
17. Sheng, Q.Z.; Qiao, X.; Vasilakos, A.V; Szabo, C.; Bourne, S.; Xu, X. Web services composition: A decade's overview. *Inform. Sci.* **2014**, *280*, 218–238.
18. Dustdar, S.; Schreiner, W. A survey on web services composition. *Int. J. Web Grid Serv.* **2005**, *1*, 1–30.
19. Rao, J.; Su, X. A survey of automated web service composition methods. In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, CA, USA, 6 June 2004.
20. Srivastava, B.; Koehler, J. Web service composition-current solutions and open problems. In Proceedings of ICAPS 2003 Workshop on Planning for Web Services, Trento, Italy, 10 June 2003; pp. 28–35.
21. Milanovic, N.; Malek, M. Current solutions for web service composition. *IEEE Internet Comput.* **2004**, *8*, 51–59.
22. Ter Beek, M.; Bucchiarone, A.; Gnesi, S. Web service composition approaches: From industrial standards to formal methods. In Proceedings of the Second International Conference on Internet and Web Applications and Services (ICIW'07), Mauritius, 13–19 May 2007; pp. 15–15.

23. Ivanova, E. Orchestrating Web Services–Standards and Solutions. In Proceedings of National Scientific Conference "Mathematics, Informatics and Computer Sciences"-St. Cyril and St. Methodius University of Veliko Tarnovo, Veliko Tarnovo, Bulgaria, 12–13 May 2006; pp. 137–142.
24. Workflow Management Coalition (WfMC) Process Definition Interface–XML Process Definition Language (WFMC-TC-1025). Available online: http://www.xpdl.org/standards/xpdl-2.2/XPDL%202.2%20(2012-08-30).pdf (accesses on 17 October 2014).
25. Van der Aalst, W.M.P.; ter Hofstede, A.H.M. YAWL: Yet another workflow language. *Inform. Syst.* **2005**, *30*, 245–275.
26. Van der Aalst, W.M.P. Don't go with the flow: Web services composition standards exposed. *IEEE Intell. Syst.* **2003**, *18*, 72–85.
27. Zhao, P.; Foerster, T.; Yue, P. The geoprocessing web. *Comput. Geosci.* **2012**, *47*, 3–12.
28. Apache Software Foundation Apache ODE. Available online: http://ode.apache.org/ (accessed on 17 October 2014).
29. Louridas, P. Orchestrating web services with bpel. *IEEE Software* **2008**, *25*, 85–87.
30. Montesi, F. Jolie. Available online: http://www.jolie-lang.org/ (accessed on 17 October 2014).
31. University of Texas at Austin Orc Language Project. Available online: https://orc.csres.utexas.edu/ (accessed on 17 October 2014).
32. Montesi, F.; Guidi, C.; Zavattaro, G. Service-oriented programming with Jolie. *Web Serv. Found.* **2014**, doi:10.1007/978-1-4614-7518-7_4.
33. Choi, Y.; Garg, A.; Rai, S.; Misra, J.; Vin, H. Orchestrating computations on the world-wide web. In Proceedings of Parallel Processing: 8th International Euro-Par Conference, Heidelberg, Germany, 27–30 August 2002.
34. Apache Software Foundation Apache CXF. Available online: http://cxf.apache.org/docs/java-to-wsdl.html (accessed on 17 October 2014).
35. Kiehle, C.; Heier, C.; Greve, K. Requirements for next generation spatial data infrastructures-standardized web based geoprocessing and web service orchestration. *Trans. GIS 2007* **2008**, *11*, 819–834.
36. Friis-Christensen, A.; Ostländer, N. Designing service architectures for distributed geoprocessing: Challenges and future directions. *Trans. GIS* **2007**, *11*, 799–818.
37. Schaeffer, B. Towards a transactional web processing service. In Proceedings of the GI-Days, Münster, Germany, 16–18 June 2008.
38. Weiser, A.; Neis, P.; Zipf, A. Orchestrierung von OGC Web Diensten im Katastrophenmanagement am Beispiel eines Emergency Route Service auf Basis der OpenLS Spezifikation. *GIS-Zeitschrift für Geoinformatik* **2006**, *9*, 35–41.
39. Sonnet, J. *OWS 2 Common Architecture: WSDL SOAP UDDI*; Open Geospatial Consortium Inc.: Wayland, MA, USA, 2004.
40. OpenPlans WPS Processes-GeoServer 2.5.x User Manual. Available online: http://docs.geoserver.org/stable/en/user/extensions/wps/processes.html#process-chaining (accessed on 24 June 2014).
41. Taverna Project Home Page. Available online: http://www.taverna.org.uk/ (accessed on 12 May 2014).
42. De Jesus, J.; Walker, P.; Grant, M.; Groom, S. WPS orchestration using the Taverna workbench: The eScience approach. *Comput. Geosci.* **2012**, *47*, 75–86.

43. Brauner, J.; Foerster, T.; Schaeffer, B.; Baranski, B. Towards a research agenda for geoprocessing services. In Proceedings of the 12th AGILE International Conference on Geographic Information Science (2009), Chicago, IL, USA, 24–28 August 2009; Volume 1, pp. 1–12.
44. Menascé, D.A.; Mason, G. QoS issues in web services. **2002**, doi:10.1109/MIC.2002.1067740.
45. Spark—A Small Web Framework for Java. Available online: http://www.sparkjava.com/ (accessed on 14 May 2014).
46. openRDF.org. Available online: http://www.openrdf.org/ (accessed on 14 May 2014).
47. Müller, M.; Bernard, L.; Kadner, D. Moving code—Sharing geoprocessing logic on the web. *ISPRS J. Photogramm. Remote Sens.* **2013**, *83*, 193–203.
48. Fowler, M. *Domain Specific Languages*, 1st ed.; Addison Wesley Professional: Boston, MA, USA, 2010; pp. 105–111.
49. Xtext-Language Development Made Easy. Available online: http://www.eclipse.org/Xtext/ (accessed on 14 May 2014).