A Client for Distributed Geo-Processing on the Web

Theodor Foerster[1] and Bastian Schäffer[2]

[1] International Institute for Geo-Information Science and Earth Observation
P.O. Box 6, 7500AA Enschede, the Netherlands
foerster@itc.nl
[2] Institute for Geoinformatics
Robert-Koch-Str. 26-28, D-48149 Münster, Germany
schaeffer@uni-muenster.de

Abstract. The OGC Web Processing Service specification provides a means to perform web-based processes on distributed geodata and to produce thereby web-based geoinformation. However a client which integrates different data and geo-processing services is missing but would enable fully web-based information integration. This was the starting point for this study to build a client, which is able to integrate multiple web-based data and processing services. The applicability of the client for web-based information integration will be demonstrated for a real-time risk management scenario. The client is realized on top of uDig and available under Open Source license (GPL) at the 52° North Open Source initiative.

Key words: Web Processing Service, Web Processing Service Client, web-based processing

1 Introduction

As computational power and network capabilities mature, processing of distributed data towards information becomes one of the main interests in the IT world. This aspect is addressed by the evolving concept of Web Services and Service-oriented Architecture (SOA, [1]). Distributed processing using Web Service technology is also a main interest in the geoinformation (GI) domain, as most of the GI applications involve large amounts of globally distributed data and as the demand for distributed available geo-information increases. The web-based geodata access was addressed by the concept of Spatial Data Infrastructures (SDI, [2]) and the invention of Web Service and geodata standards as for instance developed by the Open Geospatial Consortium (OGC[3]). Web-based geoinformation has been identified as a key factor for SDIs in the future, therefore sufficient concepts for web-based processing are required. Such processes have to be able to access globally distributed data and to provide the information in line with the already available standards. Regarding such a standardized

[3] OGC website: www.opengeospatial.org

service for web-based processing the OGC proposed the Web Processing Service (WPS) specification as a discussion paper[4] [3].

In the last two years several communities started to implement this specification [4, 5] and demonstrated its applicability by several use cases, such as generalization processing [6] or groundwater vulnerability measurement [7]. We developed a processing client based on the *Java Uniform Mapping Platform* (JUMP[5]) [6]. Additionally [8] built a small demo application, but his demo did not provide flexible access to WPS nor integration with other services. So a flexible client application, which utilizes the capabilities to process distributed data services over the web and present the result in a pleasing way to the user has been missing. This was the starting point for this study to investigate the benefits of such a client and to implement a ready-to-use client solution. We decided to implement it as a desktop application as such a breed of application provides more flexibility to the user such as advanced data querying and visualization capabilities. We chose the *User-friendly Desktop Internet GIS* (uDig[6], [9]) like [8] already did for a small demo application on WPS as the appropriate implementation platform, because it already allows connecting to distributed data services such as Web Map Service (WMS) and Web Feature Service (WFS). Thereby it provides many advantages against JUMP, which does not allow to connect to any OGC Web Service and which only provides limited styling capabilities to the user. The client is realized as a plug-in for uDig and is available under Open Source license (GPL) at the 52° North Open Source initiative[7]. It is important to note, that the presented work is unique in the evolving field of web-based geo-processing, as it focuses on a flexible processing client, which allows integrating data and geo-processing services.

In Section 2, the paper describes the technical background of WPS and the idea of processing distributed services as utilized in the proposed client. Then we provide in Section 3 some insights into the client architecture. Section 4 demonstrates a real-time risk management scenario incorporating generalization functionality and buffer analysis. Both processes demonstrate the applicability of the client and the benefit of distributed data processing. Finally we discuss some problems we encountered during this study and give an outlook about possible extensions of the client. We also draw a conclusion about the practical experience with the uDig client in comparison to JUMP. The paper ends with a conclusion.

2 Technical Background

The Web Processing Service specification describes a way how to publish and perform processes on the web. Such a process can range from a simple buffer calculation to a complex process of vector analysis for generalization purposes.

[4] The discussion paper is known as WPS version 0.4.0.
[5] JUMP website: `www.jump-project.org`
[6] uDig website: `udig.refractions.net`
[7] 52° North website: `www.52north.org`

The communication is based on three operations (Figure 1), which can be called via HTTP-GET and key-value pair (KVP) encoding or HTTP-POST and a XML-encoding. The *GetCapabilities* operation (step I. in Figure 1) provides service metadata and some brief description of the process. The process metadata including the parameter descriptions is accessible through the *DescribeProcess* operation (step II.). Via the *Execute* operation it is possible to call the desired process (step III.).

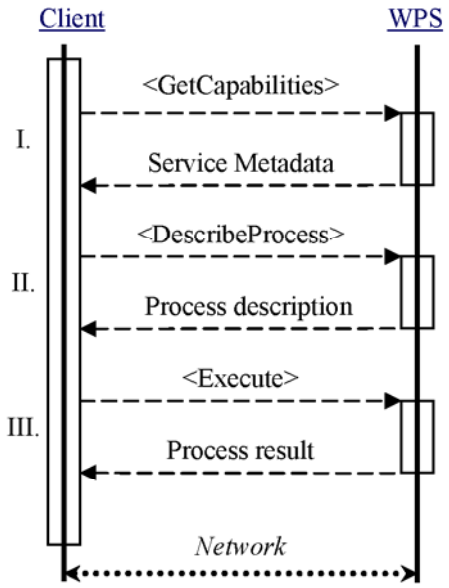

Fig. 1. Basic client-service communication pattern to perform a desired process on WPS.

Besides capabilities to perform long-term processes and to store process results on service side, WPS provides the capability to link distributed resources into the process via an Uniform Resource Locator (URL). This URL might identify a location at which a GML file is stored or even be a WFS call for a specific feature type. This reference will be incorporated in the Execute request to the WPS as a *ComplexValueReference*[8] element (Example 1).

Example 1 (Sample ComplexValueReference element linking WFS road data).
<wps:ComplexValueReference ows:reference="**http://geoserver:8080/geoserver/wfs?SERVICE=WFS&VERSION=1.0.0&REQUEST=GetFeature&typename=roads"** schema="http://schemas.opengis.net/gml/2.1.2/feature.xsd"/>

[8] This paper is based on the version 0.4.0 of the WPS specification, so the name of the ComplexValueReference element might differ in the coming versions.

By using this ComplexValueReference element, the client does not have to take care about sending the plain data to the WPS, but only the reference. Additionally such references provide a certain degree of freedom to the WPS implementation about how to retrieve the data or enable even to apply some caching mechanism based on comparing the references over multiple requests. Overall this feature improves the performance of web-based processes on client and service side drastically, as it decreases the workload to prepare the data for sending at the client side and as it enables service-side caching as already mentioned. Additionally looking at the topology of the network, most services are able to retrieve data faster than clients can send them, because services are mostly located in faster network environments.

Besides improved performance linking resources via ComplexValueReference incorporates the notion of distributed data access, as the WPS is able to incorporate multiple data from different locations. This distributed notion guarantees real-time data access. However from the perspective of caching, this might cause some trouble, as some data will be cached but has been updated at the original source in the meanwhile. So a trade-off will always remain between caching (i.e. for preformance purpose) and real-time data processing. Section 4 including Figure 7 demonstrates the application of processing referenced data services in WPS.

3 Client Architecture

The client is realized as a plug-in in uDig, which is organized as a set of plug-ins on top of the eclipse rich client platform (RCP[9]). The eclipse RCP implements the concept of inversion of control and allows thereby only specific parts of the workflow to be customized. The customization within the eclipse RCP is achieved via so called extension points, at which the intended functionality can be inserted into the RCP framework. The concept of extension points is inherited by uDig to insert intended functionality at designated points of the uDig workflow.

So in order to register a new service type, like the WPS, in uDig, the `net.refractions.udig.catalog.ServiceExtension` extension point and the `net.refractions.udig.catalog.ui.ConnectionFactory` extension point have to be implemented. The `ServiceExtension` allows the introduction of new service types to uDig. The `ConnectionFactoy` requires a custom wizard and a class extending `net.refractions.udig.catalog.ui.UDIGConnectionFactory` as a binding between the user interface and the internal model of uDig. The simplified architecture is presented in Figure 2.

The `WPSServiceExtension` is a `net.refractions.udig.catalog.ServiceExtension`, which is the major hook to make a WPS available as a new datasource within uDig. The class acts mainly as a factory to create new `WPSServiceImpl` instances, which inherit from `net.refractions.udig.catalog.IService`. Such an instance will only be created, if the entry URL of the WPS instance is

[9] eclipse RCP website: `www.eclipse.org/rcp/`

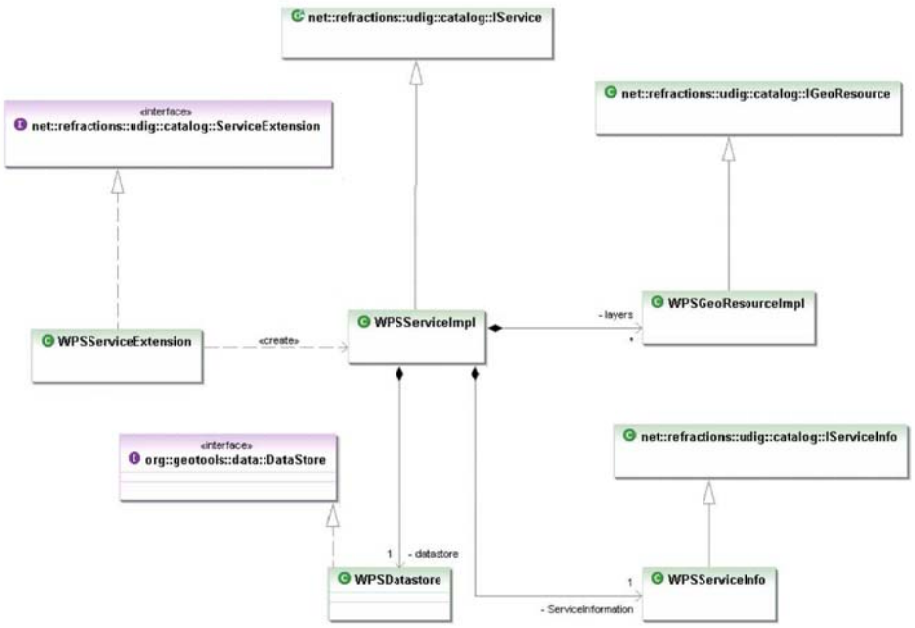

Fig. 2. Simplified class diagram of the developed uDig plug-in.

not yet available in uDig. Meta information about the represented WPS such as the URL, the description and the name are stored in `WPSServiceInfo`, which inherits from `net.refractions.udig.catalog.IServiceInfo`. The WPS does not provide layers like most of the common OGC Web Services. Since uDig applies this common OGC Web Services layer concept, the plug-in has to provide the process results as layers. Therefore, the process results are available to the layer view of uDig through `WPSGeoResourceImpl`.

The integrated execution of a WPS process and the rendering of the process results as layers in uDig involves several steps. This procedure goes beyond the current wizard concept of uDig, because it is mainly designed for a simple datasource import, which is handled by a single wizard page. Therefore the concept of wizards in uDig was extended by enabling the wizard to consist of more than one custom page. This allows separating the complex WPS execution procedure into smaller steps.

Overall, the basic interaction between the user and uDig is realized as follows (Figure 3): First the designated type of service has to be chosen - in this case WPS (step I.). This kicks-off the previously mentioned `WPSConnectionFactory`, since it was registered with the WPS service. The `WPSConnectionFactory.createConnectionpage()` method creates and returns the first custom wizard page to the enclosing wizard (step II). The page requests the user to enter the entry URL to the WPS. After the user has pressed the wizard's next button, the client requests automatically the capabilities for the entered URL and for each of the processes the process description. Thereupon, the second wizard page is displayed and now the user can select the process (step III.), he/she wants to use. The selected process is internally stored and after pressing the next button, the third custom wizard page is shown. This page allows the user to configure the selected process with the necessary parameters (step IV.). The form for the parameters is generated on-the-fly, based on the process description. For every required literal input, the user is presented with a corresponding text box. Additionally, for every complex input, the user can choose from a drop down list containing currently available layers inside of uDig. If the user chooses a WFS layer, the layer can be send by reference according to the general wizard configuration. This allows the WPS to fetch the data and saves up time, since not the whole data has to be prepared on the client-side and transferred to the service (Section 2). In the last step (step V.), the wizard shows a final layer selection wizard page, which allows the user to select the layer(s) he/she wants to add to uDig.

To get more insight into the internal workflow of the client and how the classes cooperate to perform according to the user's action in the wizard, the remainder of this section will summarize the course of action in more detail. So after the user has pressed the next button in step IV., uDig recognizes that there are no more custom wizard pages provided and tries to fetch all layers that can potentially be added. Therefore, a `WPSServiceImpl` instance is created through the `WPSServiceExtension.createService()` method. Figure 4 presents the course of action taken to get all potential layers.

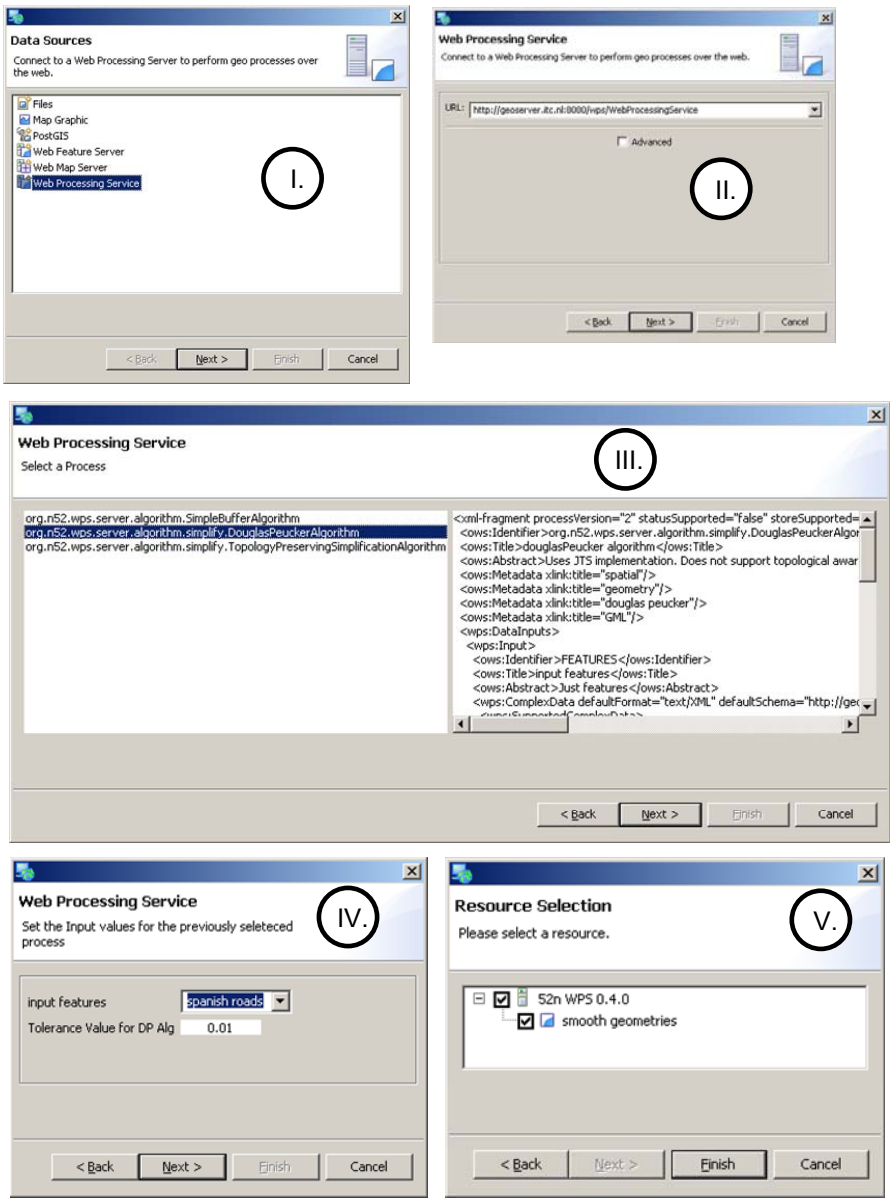

Fig. 3. The sequence of wizard pages to add a WPS process in uDig. I.) Select service type, II.) Enter WPS entry URL, III.) Select process, IV.) Configure process parameters, V.) Select process output.

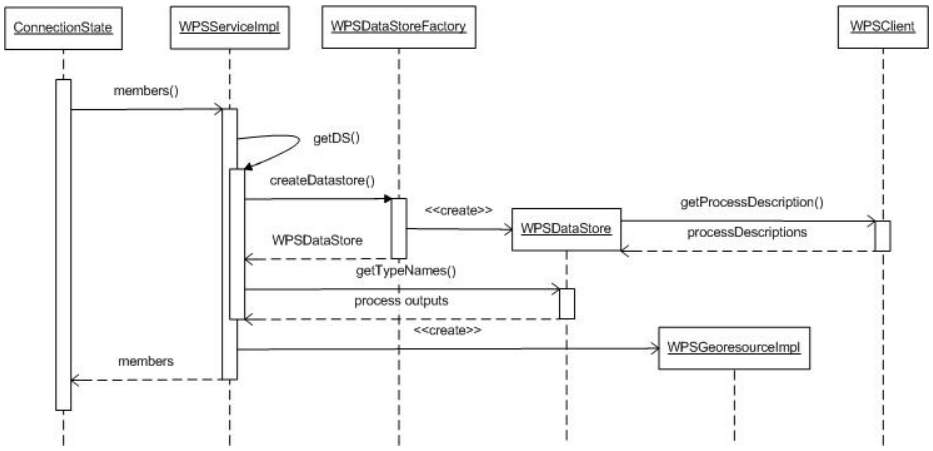

Fig. 4. Sequence diagram describing the creation of uDig layers representing WPS results.

First, the wizard, represented by the `ConnectionState`, calls the `members()` method on the `WPSServiceImpl` instance. This method obtains a `WPSDataStore` instance through the `WPSDataStoreFactory` class. The `WPSDataStore` receives all parameters collected during the previous wizard pages and fetches all process descriptions from the 52° North WPS Client API, represented by the WPSClient class in Figure 4. Next, the `WPSServiceImpl` instance calls the `getTypeNames()` method on the previously created `WPSDataStore` instance. This method analyzes each stored process description and returns the name for each complex output from the requested process. Now, the `WPSServiceImpl` instance is able to create a `WPSGeoResourceImpl` object for each returned process output, which is finally returned back to the calling `ConnectionState` object.

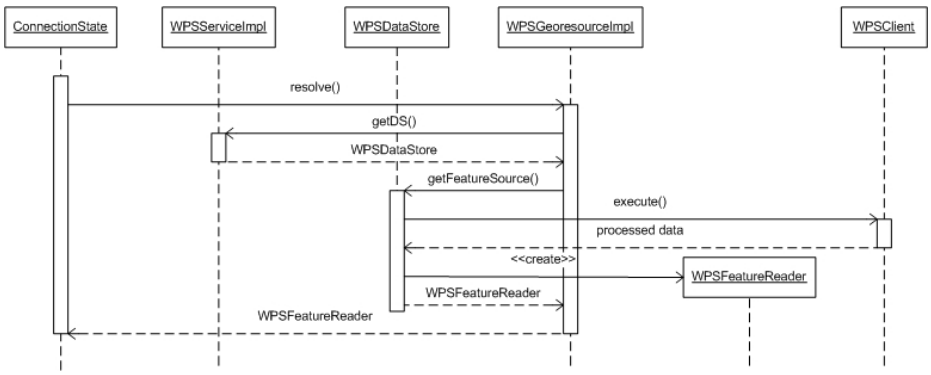

Fig. 5. Sequence diagram describing the execution of a WPS process inside of uDig.

The wizard page in step V. is based on the results returned from the `WPSServiceImpl.members()` method. Finally, the user has to press the finish button to start the actual WPS process execution, which is illustrated in Figure 5. For each selected layer, the corresponding `WPSGeoResourceImpl` object is requested to return the associated features. Therefore, the `resolve()` method is called, which obtains the `WPSDataStore` from the corresponding `WPSServiceImpl` object and fetches the features from the `WPSDataStore` instance by calling the `getFeatureSource()` method. This method constructs the actual WPS execute request based on the stored parameters collected during the wizard page process and sends the request to the `WPSClient`, which finally forwards it to the WPS over the wire.

The process results are cached and passed into a newly created `WPSFeatureReader` object, which parses the data and makes it available. The `WPSFeatureReader` object is returned back to the `WPSGeoResource` and finally forwarded to the calling `ConnectionState` object to be displayed in uDig. As a result, a new layer is added to the layer list and the associated features are displayed. From now on, this layer can be treated as any other uDig resource and especially serve as input for another WPS process.

4 A Distributed Processing Scenario

The scenario aims at producing a readable map to indicate recent fire threats to transport infrastructure. Thus this scenario involves data about fire threat areas (i.e. areas where fire has been reported) and road data. It inherits aspects of a real-time risk-management application, as different data source have to be integrated and processed in order to improve decision making. The chosen location for this scenario is the North-West of Spain. The data for the burnt areas are provided by the courtesy of the Joint Research Center in Italy and served through a WFS. The roads of Spain are taken from CORINE data, provided and served by ITC's WFS. In order to produce a satisfying map we also incorporated some CORINE landcover data, served through a WMS.

However these data sources do not contain the required information per se. In order to analyze, which roads are at stake by such a fire threat, buffers are created around the burnt areas. Additionally, as the roads are too detailed to be displayed on a general overview map, simplification has been applied to the roads. Overall, this involves two processes: Buffering the burnt area data and simplifying the road geometries. Both processes are available through the standardized WPS interface. An overview of the involved services, their data and processes is depicted in Figure 6. The WFS instances are geoserver[10] implementations and the WPS instance is based on the 52° North implementation of the WPS. The result of the described scenario with some CORINE data as background information (served through a WMS) is presented in Figure 7.

It is important to note, that this scenario is completely based upon distributed services. Especially incorporating the burnt areas data demonstrate the

[10] geoserver website: `www.geoserver.org`

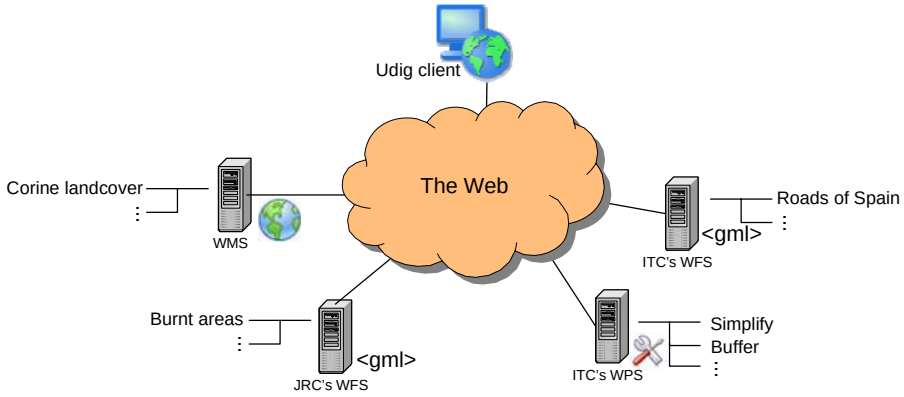

Fig. 6. Overview of the involved services in the described scenario.

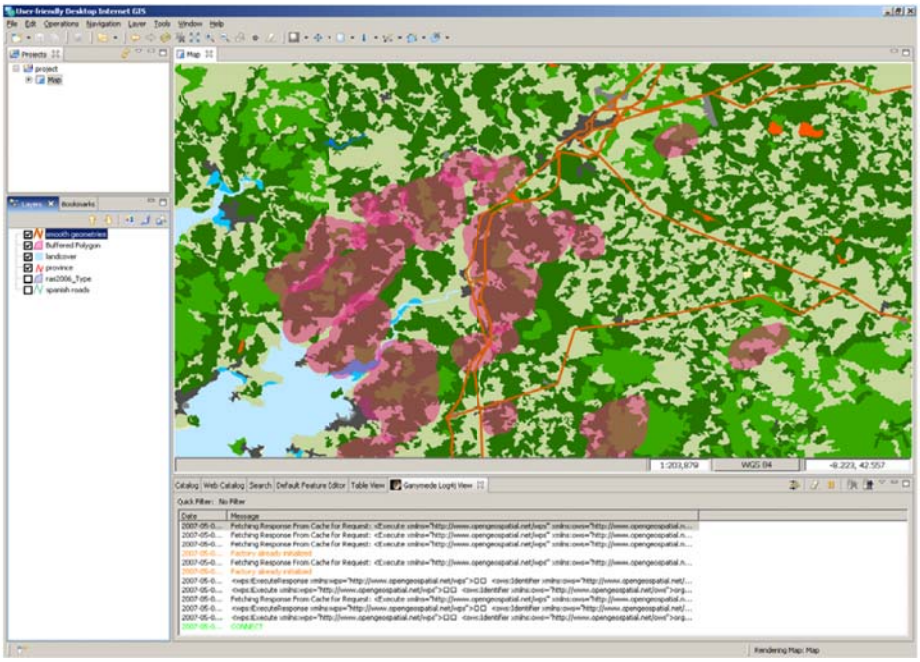

Fig. 7. Screenshot of the final result as described in the scenario.

applicability of this client for real-time data processing and visualization in a risk management scenario. Such burnt areas could be collected from sensors and published on-the-fly to a WFS, as it is described in [10] for a fire alert information system in South Africa. Thus, using ComplexValueReferences allows us to dynamically process real-time data (from WFS) and extract real-time information (by WPS).

The client resolves the references of the data to be processed automatically and incorporates them into the Execute request just as depicted in Example 1 of Section 2 (and Figure 8). Therefore the user has to register the different services into the client by indicating the endpoints of the suggested services. Based on this manual registration the client is able to retrieve the metadata of these services (via GetCapabilties & DescribeProcess) automatically. Afterwards the client is able (based on additional parameters for the processes) to perform a sequence of processes as proposed in this scenario and depicted in Figure 8.

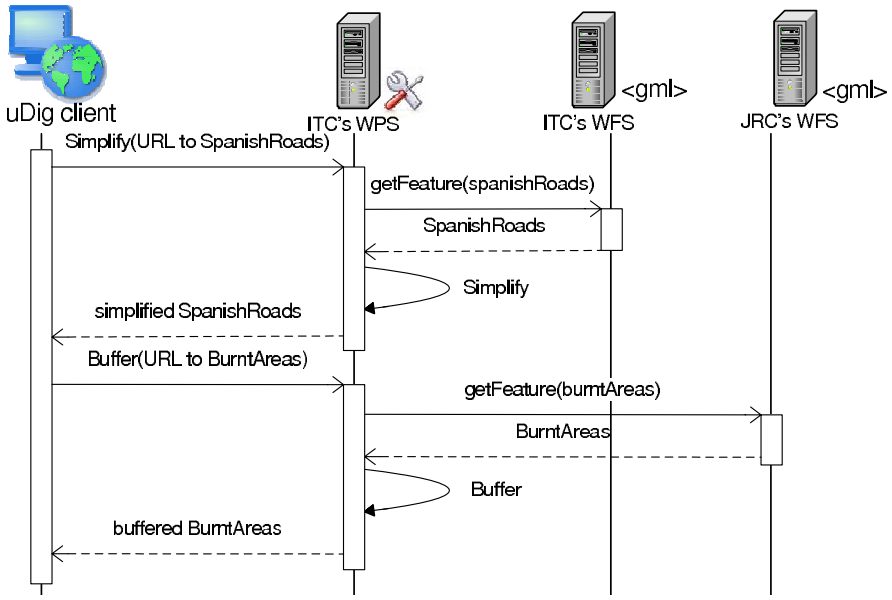

Fig. 8. Sequence diagram of the applied web-based processes (simplifying & buffering) incorporating the effect of automatically referencing the distributed data sources (spanish roads & burnt areas).

5 Outlook & Conclusion

During the implementation and the testing we got a lot of insights into the WPS specification and the architecture of uDig. While uDig is in principal a real powerful tool, the concept of inversion of control can sometimes limit its capabilities. Comparing finally our experiences with JUMP [6] with our recent experiences gained from uDig, it becomes clear that uDig's capabilities to access data services eased the development significantly. In the case of JUMP we would have had to implement the connection to the data services from scratch. So this was a big advantage of uDig. Also the visualization part is more pleasing than in JUMP. However the developing effort of the wizard pages in uDig to configure the connection to the WPS in a user-friendly and flexible way was significantly higher, than it would be in JUMP, due to the limitations caused by the concept of inversion of control (details see in Section 3).

The WPS has also demonstrated the powerful feature of referencing distributed services. However some aspects of the specification are still not really sufficient. For instance, it is not possible to handle typed references. Such references would indicate the type of service is referenced in the ComplexValueReference and would thereby enable the WPS to apply different strategies of retrieving and caching the data.

In a further stage of the research we want to incorporate more service chaining capabilities in the client to model, deploy and trigger complex chains of processing and data services based upon WPS and BPEL.

Overall this study presents a client which can be applied for information integration in a real-time risk management scenario based upon a distributed service architecture as proposed in the Orchestra Project [11] or in the AFIS project [10]. The scenario demonstrates that processing is required to extract the required information (e.g. buffered area, simplified road geometries). Udig has proven to be a suitable platform to integrate data and processing services automatically in a user-friendly way.

The presented scenario in this paper is described in a hands-on tutorial and demonstrated in a screencast. Both plus the ready-to-use uDig plug-in are available through the 52° North website.

Acknowledgements

We highly appreciate the contribution of Dr. Rob Lemmens and Barend Köbben (both colleagues at ITC), who supported us presenting our work at the AGILE workshop 2007 about a Test-bed for geospatial web service interoperability. We also want to thank the Joint Research Center (JRC) for providing the data for the scenario. The work of Theodor Foerster is covered by the RGI 002 project[11].

Finally we want to thank the anonymous W2GIS reviewers for their constructive remarks.

[11] RGI 002 project website: `www.durpondergronden.nl`

References

1. Gottschalk, K., Graham, S., Kreger, H., Snell, J.: Introduction to web service architecture. IBM Journal **41**(2) (2002) 170–177
2. McLaughlin, J., Groot, R.: Geospatial data infrastructure : concepts, cases and good practice. Spatial Information Systems and Geostatistics Series. Oxford University Press (2000)
3. OGC: OpenGIS Web Processing Service. OGC discussion paper OGC 05-007r4, Open Geospatial Consortium (2005)
4. Foerster, T.: An open software framework for web service-based geo-processes. In: Free and Open Source Software for Geoinformatics, Lausanne, Switzerland (11-15 September 2006)
5. Cepicky, J.: Grass goes web: PyWPS. In: Free and Open Source Software for Geoinformatics, Lausanne, Switzerland (11-15 September 2006)
6. Foerster, T., Stoter, J.: Establishing an OGC web processing service for generalization processes. In: ICA workshop on Generalization and Multiple Representation. (2006)
7. Kiehle, C.: Business logic for geoprocessing of distributed data. Computers & Geosciences (2006)
8. Johansson, J.: Standardized access to geospatial information and services. Master thesis, Department of Computing Science, Umea University, Sweden (Spring 2006 2006)
9. Ramsey, P.: udig desktop application framework. In: Free and Open Source Software for Geoinformatics, Lausanne, Switzerland (11-15 September 2006)
10. McFerren, G., Roos, S., Terhorst, A.: Fire Alerts for the Geospatial Web. In: The Geospatial Web. Springer (2006)
11. Annoni, A., Bernard, L., Douglas, J., Greenwood, J., Laiz, I., Lloyd, M., Sabeur, Z., Sassen, A.M., Serrano, J.J., Uslaender, T.: Orchestra: Developing a unified open architecture for risk management applications. In van Oosterom, P., Zlatanova, S., Fendel, E.M., eds.: Geo-information for Disaster Management. (2005)