Priority enabled Web Processing Service

Final report - 52°North Student Innovation Prize 2009

Thorsten Deelmann (th.deelmann@wwu.de)

Martin Wilden (martin.wilden@wwu.de)

Introduction

Spatial Data Infrastructures are focused on allocation of data in a standardized way. Geoprocessing services came into it to gain information from the data. This becomes more and more interesting in GIS-environments to gain information from geospatial data in an efficient and interoperable manner. Often, processing of geospatial data need huge among of memory and CPU resources. Unfortunately it is not possible to prioritize processes of a WPS, so that more important calculations have precedence. Furthermore the user of the service is not able to control his processes (pause, resume or stop). It also would be useful to see which processes are running and which jobs are in the queue. The WPS would become much more flexible and user-friendly. Beyond this, new business models can be reached, that provide different kinds of processes and priorities as an external attendance.

Concept

In general a WPS is a standardized interface, which should make it easier to use geospatial processes. A process contains an algorithm working on geospatial data. For OGC-compliant WPSs 3 Operations have to be implemented:

- GetCapabilities
- DescribeProcess
- Execute

As the base of our work, we use the 52°North WPS. It has the ability to work off multiple Jobs simultaneously. Jobs, which cannot be executed, are put into a queue. The only criterion is the submission-time, so that we have the principle “First come, first serve”. This is insufficient in many use-cases. So we define interfaces, which allow a prioritization of jobs. Therefore the WPS is extended with the following operations:

- GetJobs
- PauseJob
- CancelJob
- ResumeJob

An important factor is the issue of security. Only the original submitter of the job should be authorized to stop, pause or resume a job. The OASIS-standard “Web Service Security” (WS-Security) gives a possibility to exchange identity information. For the Execute-operation and all new operations (except GetJobs) a WS-Security-compliant “SecurityToken” is required. The SecurityToken

will be extended so that it optionally includes a priority element. This is needed for the Execute-Operation.

Out of the SOAP exchange protocol multiple specifications in the area of web services – called WS*-framework – have been developed. The exchange of SecurityTokens is specified in WS-Trust. Therefore a “Security Token Service” (STS) was established, which we use in our architecture.

So how does the client know that he needs this SecurityToken? When sending a GetCapabilities-request to the WPS, the client gets the information that a SecurityToken is required for authentication of the user. The client calls a STS (already implemented in the security community of 52°North) and gets the SecurityToken if the authentication was successful.

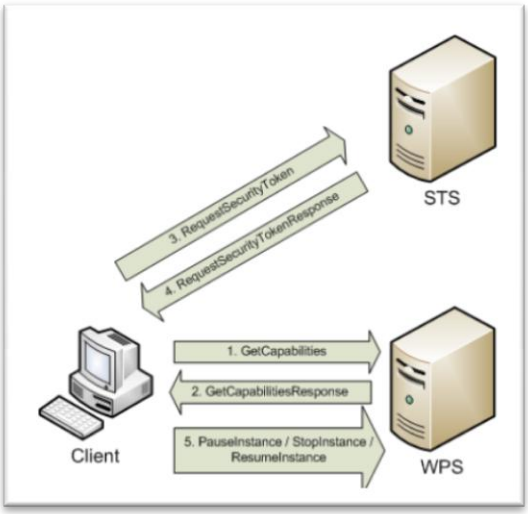

Figure 1: External workflow

For all existing operations the WPS has a HTTP-SOAP-interface. Regarding the new operations we also want to use the SOAP protocol. A SOAP-message consists of a Header (optional) with metadata and a body with the payload. So far SOAP-messages are sent with an empty Header. This will change introducing the SecurityToken. The advantage is that the Execute-request-schema stays untouched, because only a Header is added to the request.

Figure 2: Makeup of the SOAP-Message including the PriorityToken

Results

The following subsection shows problems, which occurred and how we solved them.

Problem: How to persist Jobs?

One of the main questions is how to persist paused Jobs of the WPS. This becomes important because it would not make any sense to hold intermediary results in the RAM of the machine on which the WPS is running. The consequence would be a slowdown of the WPS and at any time the service would crash. So the goal was to store intermediate data on the hard drive to recover the data at the point the job is resumed.

Considered Solution: Serialization

By taking a look at the technology of the WPS, every new Process is related with a new Thread. This implicates, that persisting a Job is equal to serialize a Thread. But this is simply not possible in Java. So we investigated for other possibilities of how to make the results persistent. One thought is to make every Algorithm serializable. This would be helpful because there is a 1:1-relationship between a Process and an Algorithm. The disadvantage of this approach is that the WPS-P would be a less generic version of a WPS. Every new Algorithm has to implement the Serializable-Interface, so that the Algorithm would eventually not work on another WPS.

Problem: Job information

As already mentioned the Owner, the Priority and the Submission Time of a Job are sent in the SOAP-Header of a request. So the task is to get this information out of the SOAP-Message and bring it together. The retrieved Metadata has also to be assigned to a correspondent WPS process.

Solution: Metadata population with SOAP-Header-Processor

At client side the request has to contain the correspondent Metadata. We implemented a SOAP-Header-Processor, which gathers the relevant information at server side. Owner, Priority and Submission Time are then stored in a JobMetadata-Object, which is just a kind of POJO to hold the data. Such an Object is assigned to every WPSTask.

Problem: Job organization

Incoming Jobs have to be arranged in a way to have access to their information and to control them. This means, that you have the ability to look at a status of a Job and if applicable change the status. To identify a Job in a request, every Job must get a unique ID.

Solution: Priority Registry

To get control of the Jobs on technological level, we implemented a data structure called PriorityRegistry. This is basically a ConcurrentHashMap in which Objects of the type PriorityRegistryEntry. Beside the populated Metadata such an entry has additional attributes “status”

and “executionFinishedTime”. It also holds the Runnable to start a Thread. The unique ID for every Job is generated by a hash function.

Internal Workflow

The internal workflow consists of five components. A kind of dispatcher is responsible to forward requests to the correspondent handler. This handler uses the SOAPHeaderProcessor to extract the information from the SOAPHeader. After that, additional metadata is retrieved by the metadata populator. The PriorityRequestHandler adds the job to the PriorityRegistry and finally the PriorityRequestExecutor executes the job and modifies the PriorityRegistry.

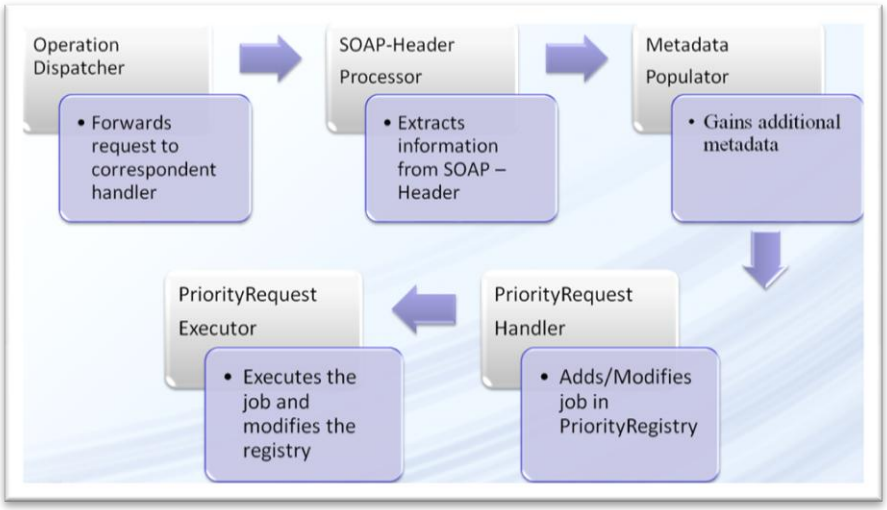

Figure 3: Internal workflow

uDig-plug-in

This is a plug-in developed by 52°North for the Open Source GIS uDig. It enables uDig to deal with WPSs. We extended the plug-in so that it can also handle the WPS-P but still the “normal” WPS. It detects if the given WPS is priority enabled by a PriorityEnabled-Element in the Capabilities Document. So we had to change the schema for the GetCapabilities-response of the WPS, too.

Lessons learned and future work

After starting the implementation it became clear that we would not be able to realize the whole amount of our ideas. We had to decide what the most important part of our work should be. So it came out that it would be realistic to setup/extend (PriorityToken) the security-infrastructure (Client, STS, WPS) and implement the process-controlling-ability.

For future work the persistence-mechanism which stores intermediary results on the hard drive has to be implemented. It is also reasonable to refine the uDig-plug-in to demonstrate a complete use-case to a broader community.

It is also imaginable to bring licensing-functionality into the architecture. This means that the user would conclude a license to have the ability to only execute requests with certain priorities. This would implicate, that the Client asks the STS only for a normal SecurityToken instead of a PriorityToken. Such a “LicenseBroker” is in the process of being implemented by the 52°North Security Community right now. But for the use with the WPS-P further implementation work on this component would be essential.

Code repository

The implementations of the 52°North WPS-P, STS and uDig-plug-in are accessible in the 52°North SVN repository:

http://52north.org/svn/geoprocessing/main/WPS/branches/WPS-P/

http://52north.org/svn/security/52n-security-apps/trunk/52n-security-sts-webapp/

http://52north.org/svn/geoprocessing/incubator/WPS-Client%20udig/branches/WPS-Client%20udig-priority/