Master Thesis

im Rahmen des Universitätslehrganges "Geographical Information Science & Systems" (UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS) der Paris Lodron-Universität Salzburg

zum Thema

"WPS for local SDIs – A case study about the applicability of web processing services (WPS) for Freiburg's spatial data infrastructure (SDI)"

vorgelegt von

Dipl.-Ing. (FH) Gunnar Ströer

591431, UNIGIS MSc Jahrgang 2016

Betreuerin:
Prof. Dr. Barbara Hofer

Zur Erlangung des Grades
"Master of Science (Geographical Information Science & Systems) – MSc(GIS)"

Gundelfingen, 24. May 2019

Acknowledgements

I would like to thank Prof. Dr. Barbara Hofer from the University of Salzburg for the supervision and the valuable feedback during the whole process of writing this thesis.

I thank in particular Michael Schulz, head of the IT department of the Freiburg i. Br. city administration, for giving me professional advice regarding technical aspects and its complex dependencies.

Also thanks to Stefan Trometer from the CADFEM company and the team around Arno Klomfass from the Fraunhofer Ernst-Mach-Institute (EMI) for making available and assistance in understanding the APOLLO Blastsimulator software.

Thanks to the developers and members especially of the PyWPS as well the OSGeo mailing lists.

Furthermore, thanks to the entire University of Salzburg UNIGIS team for their support throughout this master's program and to my employer for enabling a flexible time management and that I could write my thesis at the spatial data management department.

Finally, I would like to acknowledge the patience of my family and friends who supported me in writing this thesis.

Science Pledge

I certify by my signature that this thesis is entirely the result of my own work and that it has not been submitted anywhere for any award. I have cited all sources of information I have used in my thesis.

Gundelfingen, 24. May 2019 — Gunnar Ströer

Abstract

The build-up of local spatial data infrastructures (SDI) has been pushed forward in the last few years, not least because of the impact of the INSPIRE directive. The approach of a Service Oriented Architecture (SOA) based on the open standards of OGC has proved its worth. At the same time, the increasing digitalization of municipal administrations is creating the need for automation of complex processes that extend into a wide range of disciplines.

The Web Processing Service (WPS) standard approved by the OGC in 2007 can be used for the implementation of processes, and has the potential to connect municipal process flows to be adapted in the sense of digitization with an SDI, and to share the advantages of an SDI with external procedures that have not yet been able to be connected. Whether the implementation and use of WPS processes is applicable and feasible for a local SDI is examined in the context of this master thesis by means of a complex and real existing use case.

The scenario of the use case includes the evacuation planning in the Explosive Ordnance Disposal (EOD). An external component for the simulation of an explosion plays a special role. A total of eight different WPS processes were implemented and chained in two different ways. The examination regarding the applicability of WPS in a local SDI is measured on the one hand by the actual implementation and on the other hand by three general criteria: reusability, compatibility and usability.

Keywords: OGC, WPS, Web Processing Service, SDI, Spatial Data Infrastructure, Service Chain, Orchestration, Freiburg, Local Authority, Evacuation, Explosive Ordnance Disposal

Kurzfassung

Der Aufbau kommunaler Geodateninfrastrukturen (GDI) wurde in den letzten Jahren, nicht zuletzt aufgrund der Betroffenheit durch die INSPIRE-Richtlinie, vorangetrieben. Dabei hat sich der Ansatz einer dienstorientierten Architektur SOA auf Basis der offenen Standards des OGC bewährt. Gleichzeitig weckt die zunehmende Digitalisierung kommunaler Verwaltungen den Bedarf an der Automatisierung auch komplexer und in verschiedenste Fachdisziplinen hineinreichende Prozessabläufe.

Der 2007 durch das OGC verabschiedete Web Processing Service (WPS) Standard kann für die Implementierung von Prozessen herangezogen werden, und hat das Potenzial kommunale, im Sinne der Digitalisierung anzupassende Prozessabläufe mit einer GDI zu verbinden, und bisher nicht erreichbare fachfremde Verfahren an den Vorteilen einer GDI teilhaben zu lassen. Ob die tatsächliche Implementierung und Nutzung von Prozessen auf Basis von WPS für eine kommunale GDI geeignet und machbar ist wird im Rahmen dieser Masterthesis anhand eines komplexen und real existierenden Anwendungsfalls untersucht.

Das Szenario des Anwendungsfalls umfasst die Evakuierungsplanung bei der Kampfmittelbeseitigung. Dabei spielt eine externe Komponente zur Simulation einer Explosion eine besondere Rolle. Insgesamt wurden bei der Realisierung acht verschiedene WPS-Prozesse implementiert und auf zwei unterschiedliche Weisen verkettet. Die Untersuchung hinsichtlich der Eignung von WPS in einer kommunalen GDI wird zum einen an der tatsächlichen Umsetzung gemessen, und zum anderen an drei allgemeinen Kriterien festgemacht: Wiederverwendbarkeit, Kompatibilität und Benutzerfreundlichkeit.

Schlagwörter: OGC, WPS, Web Processing Service, GDI, Geodateninfrastruktur, Prozesskette, Verkettung, Orchestrierung, Freiburg, Kommunalverwaltung, Evakuierung, Kampfmittelbeseitigung

Contents

List of Figures

List of Tables

List of Listings

Abbreviations

ABK .... Fire and Disaster Control Department
AfO .... Office of Public Order
ALKIS .... Official Real Estate Cadaster Information System

BKG .... Federal Agency for Cartography and Geodesy
BPMN .... Business Process Model and Notation
BZBE .... Consulting Centre for Building and Energy Freiburg

CityGML .... City Geography Markup Language
CSV .... Comma-separated Values
CSW .... Catalogue Service for the Web

DEM .... Digital Elevation Model
DUVA .... DV-technische Unterstützung der Volkszählungs-Auswertung

EMI .... Ernst-Mach-Institute
EOD .... Explosive Ordnance Disposal
EPSG .... European Petroleum Survey Group Geodesy
ESRI .... Environmental Systems Research Institute
ETRS89 .... European Terrestrial Reference System 1989

FOSS .... Free and Open Source Software

GeoTIFF .... Georeferenced Tagged Image File Format
GIS .... Geographic Information System
GIScience .... Geographic Information Science
GLUES ..... Glob. Assess. of Land Use Dyn., Greenhouse Gas Emis., Ecosystem Srv.
GML .... Geography Markup Language

HTML .... Hypertext Markup Language
HTTP .... Hypertext Transfer Protocol

INSPIRE .... Infrastructure for Spatial Information in the European Community
IT .... Information Technology

JSON .... JavaScript Object Notation

KVP .... Key-Value-Pair

OGC .... Open Geospatial Consortium
OGR .... OGR Simple Features Library
OSGeo .... Open Source Geospatial Foundation
OSM .... Open Street Map
OWS .... OGC Web Service

PHP .... PHP: Hypertext Preprocessor
POI .... Points of Interest

SDI .... Spatial Data Infrastructure
SLES .... Suse Linux Enterprise Server
SOA .... Service Oriented Architecture
SOAP .... Simple Object Access Protocol
SQL .... Structured Query Language
STL .... Standard Triangulation/Tesselation Language

TIFF .... Tagged Image File Format
TNT .... Trinitrotoluene (Explosive)

URL .... Uniform Resource Locator
UTM .... Universal Transverse Mercator

W3C .... World Wide Web Consortium
WCS .... Web Coverage Service
WFS .... Web Feature Service
WGS 84 .... World Geodetic System 1984
WMS .... Web Map Service
WPS .... Web Processing Service
WSDL .... Web Services Description Language
WSGI .... Web Server Gateway Interface

X3D .... Extensible 3D
XML .... Extensible Markup Language

CHAPTER 1

Introduction

1.1 Motivation

Since the foundation of the Open Geospatial Consortium (OGC) in 1994, the standardization for discovery, display, exchange and processing of spatial data in the form of web services has been promoted. In the context of the open data initiative and the INSPIRE directive, interoperable approaches for the exchange of spatial data among each other as well as with citizens and industry are increasingly finding their way into public administrations. A decisive factor here is the question of the type of spatial data. If it concerns pre-processed data, then these can be made permanently available without large expenditure by means of Web Map Service (WMS) or Web Feature Service (WFS). If, on the other hand, it concerns data that must be provided individually in a time-critical application case, then this can be realized via a processing chain based on the Web Processing Service (WPS) standard (Yoon et al., 2017).

So far WPS is mostly used on topic-specific platforms which are often operated by international research institutions and in national or regional authorities and associations, i.e. which cannot be described as broadly applicable, general services (Hofer, 2015). A widespread domain is the environmental sector, for example in the automated fire detection (Samadzadegan et al., 2013) or in flood protection (Tan et al., 2016). Another example is the coupling of Sensor Observation Service (SOS) and WPS for the online geoprocessing of monitoring data of the Water Dam Measuring Information System (TaMIS) developed by the regional water authority Wupperverband. (Stasch et al., 2018). Also the project GLUES, developed by the Technische Universität Dresden, uses a WPS for different geoprocessings. At the University of Bonn Walenciak et al. (2009) have dealt with the use of WPS in 3D SDIs. There are now several examples in which a WPS is in practical use. However, these could not be assigned to the SDI of a municipal city administration.

Providers of WPS are difficult to find, especially at the municipal level. One reason for this is the lack of registration in a Catalogue Service for the Web (CSW), which makes an efficient search more difficult, as an investigation by Lopez-Pellicer et al. (2012) showed. Another reason may be that municipal administrations are very heterogeneous in their IT structure, which is due to their wide range of tasks that has led to isolated solutions (Hogrebe, 2008). For example, they are responsible for urban planning, the cadastre, building law and in many different matters for their citizens. This is accompanied by a large number of experts from different fields, who are involved in independent procedures. These experts rarely have the GIS knowledge necessary to solve their problems. This leads to the question whether WPS can offer an added value in the communal area, if the existing heterogeneity gains a little bit in interoperability, and if users outside Geographic Information Science (GIScience) can also answer spatially complex questions qualitatively and independently. But how flexible, how manageable, how sustainable can complex processes within a local SDI be implemented by means of a WPS? These questions are open, but there are existing evaluations that indicate the potential of WPS. For example, Brennecke (2015, p. 62) came to the conclusion that especially complex geoprocessing models, which cannot be reproduced easily, can be suitable for implementation as WPS.

The planning of the evacuation of an urban district in the case of disposal of explosive ordnance from the two world wars is such a complex process, and still a topical issue. The whole process is time-critical and includes actors from different disciplines and different knowledge, for example the Fire and Disaster Control Department (ABK), the Office of Public Order (AfO) and the Office for Citizen Service and Information Management (ABI). Geodata play a decisive role here, be it for the selection of evacuation areas, the marking of critical infrastructure or the effects of detonation in the event of a disaster (Stollberg et al., 2007, pp. 239–251). The city of Freiburg is no exception, as happened last in May 2019[1]. The ABK does not work with the latest available geodata and processing methods, because their systems are not directly connected to the SDI. The orchestration of a processing chain using WPS across several institutions and systems represents a possible approach to improving the overall process. The use of WPS is therefore a possibility for linking an SDI with other spatial and non-spatial methods. On the basis of this use case it is to be examined whether a process implementation corresponding to the WPS standard meets the requirements of the actors concerned and whether parts of the developed process

1 https://www.badische-zeitung.de/freiburg/blindgaenger-in-freiburg-gesichert-aber-nicht-entschaerft-anwohner-koennen-zurueck--172959594.html (visited on 10/05/2019)

chain can also be reused for completely different questions and thus represents an added value for an municipal SDI.

1.2 Objectives and research questions

The preceding research shows that the use of WPS in municipal administrations has not yet been sufficiently investigated, although this standard can also be of relevance for municipal administrations. From this the following hypothesis is derived for this master thesis:

> *The applicability of WPS processes in a local SDI based on open standards is possible and results in a significant added value due to the reuse possible because of the standardization of WPS interfaces.*

The hypothesis is tested on the basis of the implementation of a real existing use case and evaluated according to certain criteria. In order to answer the research question, the following operational subgoals are defined:

- Definition of the responsible tasks of a local SDI. Only when the area of responsibility is known a reliable scenario can be worked out.
- Description of the technical specifications and common questions of Freiburg's SDI. In order to be able to define a concrete use case it is necessary to know the relevant specifications and conditions of the SDI.
- Definition of criteria that allow a realistic verification of the hypothesis.
- Selection of a suitable use case for the abstraction of the complexity of the real world, against which the previously defined criteria can be evaluated.
- Implementation of the use case covering operations such as data delivery and spatial processing to support the evaluation of the applicability of WPS in a local SDI.
- Evaluation of the final workflow and for a local SDI based on the selected criteria.

When answering the research question, exemplary questions from the municipal administration are taken into account. Due to the large number of possible questions within a city administration, there is no comprehensive review of all kinds of (spatial) problems. Furthermore, the importance of WPS clients and workflow engines is considered, but there is no in-depth investigation.

1.3 Methods

The first step is a literature search on already realized application examples on the basis of WPS. With this it can be estimated in which institutions and in which fields WPS are used so far, and whether there are already other city administrations using WPS.

Based on the operational subgoals of the research question, the SDI relevant topics are placed in the urban context. This includes the designation of tasks and responsibilities of Freiburg's SDI, such as the connection of procedures to the SDI or the compliance with laws, as well as the technical specifications within which the answer to the research question lies. The description of typical, municipal problems, to whose solution the local SDI contributes, shows the spectrum of spatial questions. One of these cases is used as a case study and its implementation is evaluated according to the following criteria, which form the basis for testing the hypothesis:

- Reusability
- Compatibility
- Usability

The specific use case refers to the geodata-related part in the planning of an evacuation in the case of an EOD. The geodata-related questions concern the determination of the exact location of the affected area, the buildings and addresses contained therein and the critical infrastructure. In order to meet the technical requirements, the background and the entire process of such a scenario is explained. The APOLLO Blastsimulator from the Fraunhofer Ernst-Mach-Institute (EMI) for High-Speed-Dynamics – a Computational Fluid Dynamics (CFD) software for the simulation of detonations, blast and gas dynamics – plays a special role. With this software it is possible to estimate damages in case of detonation with high precision. This tool is not a GIS, so it does not support corresponding functions or geodata formats, and is therefore a good example for a highly specialized application outside the domain of GIS. Nevertheless, it is a part of the overall process that solves a problem that can only be solved by it, and thus a part of the processing chain.

This is followed by the implementation of the WPS processes with the Python programming language. The complete source code of all processes, relevant XML requests as well as extracts from the data material can be found on the GitLab[1] repository belonging to

1 https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/

this master thesis. The finished processes are tested on a virtual server provided by the city administration via XML requests. The geodata used as input or which have been processed originate from the city administration of Freiburg and are partly open data, like the Points of Interest (POI) and the 3D city model, or not freely accessible for reasons such as privacy, like owners of buildings and their addresses. Excluded from this is the result of the APOLLO Blastsimulator, which belongs to the Fraunhofer EMI. The development of WPS processes contains many freedoms. The following details are considered in more detail:

- Atomicity
- Handling of inputs and outputs
- Synchronous versus asynchronous
- Single use and chained processes

After the completed workflow for the case study has been implemented, the results are checked for plausibility. The evaluation is based on the previously defined criteria and will answer the research question within the defined context. Furthermore, advantages and disadvantages are pointed out which result from the implementation of the processes and the use of the WPS.

1.4 Structure of the thesis

The master's thesis is divided into a total of seven chapters.

This introduction is followed by a chapter on the surrounding conditions underlying this work. The most important terms, standards and technologies are introduced and explained.

The third chapter deals with the SDI of Freiburg and highlights their tasks, technical specifications and peculiarities. At the end the criteria are defined, by which the applicability of WPS can be checked.

The fourth chapter describes the selected case study and shows potential improvements that can be expected from the implementation using WPS. A decisive step here is the schematic workflow that is required for the derivation and delimitation of all necessary processes.

The fifth chapter deals with the implementation of the previous considerations. Four case-specific and four non-case-specific processes are developed and then chained according to the selected use case.

The sixth chapter evaluates the results and the application of the WPS processes against the previously defined criteria. Other advantages and disadvantages identified during implementation are also explained.

The concluding chapter summarizes the main findings of the master's thesis with reference to the research question and gives an outlook on still open questions that can be investigated in future work.

CHAPTER 2

Context and basic principles

2.1 Local government and digitization

The increasing focus on digitization leads to dynamic change processes in municipal administrations, which result in new questions, the answers to which are becoming increasingly complex (Martini et al., 2016, p. 22). Freiburg is also strongly committed to this topic, as the current digital strategy[1] reveals. This includes topics such as transparent urban planning, open data, sensor systems, 3D city models and citizen-related themes such as Volunteered Geographic Information (VGI). Often such challenges can only be mastered in an interdisciplinary way, where departments meet that have different procedures and topics and now have to harmonize them. As a result, more and more people from different disciplines are confronted with new problems, such as spatial problems, and have to be able to deal with them. This leads to the need to simplify procedures to such an extent that they can be safely applied by the responsible actors.

One challenge here is that many of these procedures contain a special component that cannot be exchanged at will because only it can perform a particular task, like cemetery management software or traffic control systems. Such components often have poor general interoperability, use other or no standards at all, and in the worst case are not compatible with the applications integrated in the planned process, so that a workaround must be found. Here are two real-life examples from everyday life for illustration:

1. Parking guidance system as real-time map: The technology used in Freiburg is based on a proprietary traffic control system with no spatial reference. The real-time number of free parking spaces is recorded per car park and collected on an external

1 `https://www.freiburg.de/pb/,Lde/1233888.html` (visited on 18/03/2019)

server. This feeds the data every few seconds into a spatial database in which the geometries are appended. A WMS extracts the geometries from the database and presents the data in a Leaflet map[1] on the municipal website.

2. Data maintenance of social institutions: The maintenance of daycare facilities is carried out in an information system called DUVA. Only an indirect spatial reference in the form of an address exists. For the representation on a digital map the periodic preparation of the CSV data in a GIS would be necessary. A solution in the sense of digitization uses the geocoding service of the Federal Agency for Cartography and Geodesy (BKG) within a script and visualizes the result with a WMS, which can be converted into an interactive map[2] on the municipal website.

The two examples show only a small part of the broad spectrum of digitization and are relatively easy to implement. The processes developed for this are proprietary, work only for the intended purpose and are not reusable for other questions. But what does it look like if a much more complex issue is to be automated? In municipal administrations, there is a wide range of tasks and thus processes. Especially in Germany, the digitization of these processes and their user-friendliness is lagging behind, although in many cases the automation and digitization has the potential to increase the quality and quantity of an authority's work (Martini et al., 2016).

2.2 Spatial data infrastructures

SDIs can be an efficient basis to support digitization, as they aim to provide spatial information to a large number of users. The share of spatial information in municipal administrations is considered high. The exact quantification of this share is difficult to prove and has settled in the industry at 80 %. A scientific study came to a share of 57 %, but experience shows that this share is higher for municipal data records (Hahmann et al., 2012). Geodata are an important part of our society today and play an important role when it comes to deciding where or where to go, for example when planning a new district. This includes not only data with a direct spatial reference, which are provided with an exact coordinate, but also data with an indirect spatial reference, for example an address. In a municipal administration, a great deal of such data is recorded, processed and output.

1 http://www.freiburg.de/pb/,Lde/231355.html (visited on 19/03/2019)
2 https://www.freiburg.de/pb/,Lde/1248538.html (visited on 19/03/2019)

An SDI is a physical network for the exchange of geodata. This data network links the different actors with each other, from the originator to the processor to the user. The aim is to establish public access to geoinformation and to reduce technical and non-technical hurdles (Altmaier et al., 2002), i.e. to increase interoperability. The structure of an SDI can be very different, ranging from a proprietary commercial one-stop solution (e.g. ESRI) to a heterogeneous architecture based on Free and Open Source Software (FOSS). From a technical point of view, the following components belong to an SDI:

- Basic geodata, which mainly come from the surveying offices, and thematic geodata, which come from the individual specialist offices.
- Metadata describing the geodata, such as source, intended use, contact person, spatial reference system or topicality.
- Geodata services that enable access to geodata, e.g. for visualization, download, research, acquisition or further processing.
- Networks, which realize the exchange at technical system level, ideally with high availability.
- Standards that ensure that communication between different components functions smoothly and guarantee a high level of interoperability.
- System-related software that makes the network accessible, such as the operating system and web server.
- Geo-related software that creates geoservices, manages geodata (spatial database), presents (web client) as well as acquires and processes (desktop client) geodata.

An SDI consists of organizational units and is subject to a legal framework that follows the long-term development of a global SDI. In Europe, the European Directive INSPIRE applies, which defines the framework for a European SDI and has an impact down to the level of a local SDI (fig. 2.1). The SDI Germany is helping to achieve these goals. The geodata affected for urban SDIs by INSPIRE include mainly the land-use plans important for urban planning. Due to the standardization, all levels can communicate with each other. Further laws at national level contribute to the formation of an SDI. One example is the German Geodata Access Act (GeoZG), which regulates access to geodata, geodata services and metadata. By harvesting mechanisms of a Metadata Information System (MIS) the metadata of other SDIs can be harvested (Kliment, 2015), whereby this happens in local

SDIs rather complementarily than in national or international SDIs, whose contents are mainly based on harvesting. The spectrum of the responsible topics and the dependencies of the underlying data models (ALKIS) tend to increase the smaller the territorial authority for which an SDI exists (fig. 2.1). And the larger the territorial authority for which an SDI is responsible, the more often metadata is harvested.

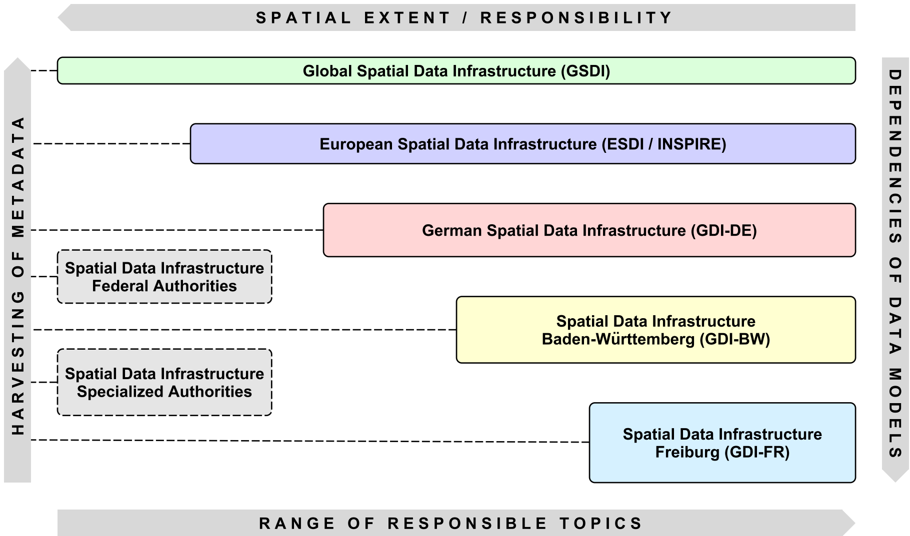

Figure 2.1: Hierarchical structure of SDIs

2.3 Interoperability by the use of standards

The term interoperability has already been used several times. Bartelme (2005, p. 363) describes interoperability as the ability to communicate, execute programs, and exchange data between functional units in a way that requires users to have little or no knowledge of the particularities of those units. To achieve this, the use of open standards is required. In the field of SDIs these are above all the standards of the OGC. The OGC has set itself the goal of advancing interoperability in GIScience and the integration of GIS in standard IT procedures (Altmaier et al., 2002). The result are services whose behavior, properties and interfaces are described by freely available specifications. The use of a Service Oriented Architecture (SOA) according to the *Publish – Find – Bind* principle is one of the essential prerequisites for interoperability. Each service supports a certain number of mandatory and optional operations. For a WMS the most common are *GetCapabilities* to describe the WMS, *GetMap* to deliver a georeferenced raster image and *GetFeatureInfo* to request object-related data for a certain position in the map. The OGC services often used in an SDI and relevant for this master thesis are briefly introduced in table 2.1.

Table 2.1: Selection of OGC standards used in SDIs

Service Name	Description
Web Map Service (WMS)	Returns a georeferenced raster map based on selected geographical layers and the area of interest.
Web Map Tile Service (WMTS)	Realizes the provision of digital maps using predefined tiles, with the goal of high performance.
Web Feature Service (WFS)	Enables access to geographic features as vector data and can manipulate geodata as a Transactional WFS.
Web Coverage Service (WCS)	Provides access to multidimensional coverage data with full semantics for machine processing.
Catalogue Service for the Web (CSW)	Publication of metadata about geo applications, geoservices and geodata in an SDI, so that they can be found.
Web Processing Service (WPS)	Spatial analysis of geodata via the Internet based on predefined processing models.

But not only services belong to the setup and operation of an SDI, also open exchange formats are essential for a high degree of interoperability. Here the XML and all formats derived from XML are of high importance. The Geography Markup Language (GML) format is widely used in geoinformatics, and Extensible 3D (X3D) as another description language based on XML has established itself in the field of 3D models. A complete overview of all OGC standards can be found on their website[1].

The use of standards increases syntactic interoperability. However, one goal of an SDI is also a semantic interoperability, which describes different subject systems according to standardized rules and does not require a uniform data model for all subject applications (Seifert, 2005). This is mainly driven by INSPIRE or the AFIS-ALKIS-ATKIS (AAA) Model of surveying administrations. Semantic interoperability is often realized via model-based transfer procedures, but this approach is not always possible in the heterogeneous structure of a municipal administration. Here the use of a WPS can also be an advantage.

2.4 Web processing services

The Web Processing Service (WPS) standard was approved 2007 by the OGC in version 1.0.0 and is available since 2015 in version 2.0. Schut (2007) defines WPS as: "A standardized interface that facilitates the publishing of geospatial processes, and the

1 https://www.opengeospatial.org/docs/is (visited on 23/02/2019)

discovery of and binding to those processes by clients." The WPS standard regulates the way a client interacts with a geoservice to perform a spatial process. The goal is a standardized interface for publishing and performing geoprocessing in a web service environment (Giuliani et al., 2012).

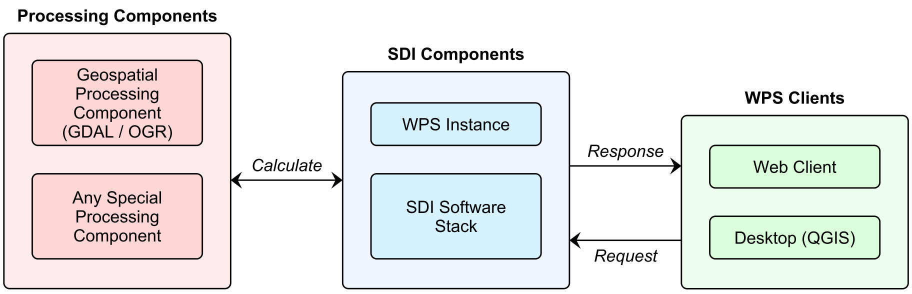

Figure 2.2: Basic principle of WPS in an SDI

In addition to the obvious benefits of availability and interoperability, there are other positive aspects, such as the generally higher performance of a server over a desktop computer, as demonstrated by Stollberg et al. (2007) in a real-time risk management scenario. A WPS can process vector and raster data, but there are no fixed bounds to the data, as shown in the case study from chapter 4. The communication between client and server should be based on XML as the preferred exchange format. The WPS specification defines three mandatory operations (fig. 2.3):

- *GetCapabilities* for the basic description of the available processes, properties and metadata of the requested WPS.
- *DescribeProcess* for the detailed description of a specific process of the requested WPS, such as inputs and outputs, metadata or supported data formats.
- *Execute* to run a specific process of the requested WPS and return the results. This operation is less specified and has low restrictions.

A WPS has a certain resemblance to WFS and WCS. Due to the large number of possible processing operations or data formats, it is clear that this standard must be particularly abstract and generically defined. The WPS standard has the following capabilities.

- Inputs can be a web-accessible URL, like from a WFS, or embedded in the request.
- Outputs can be stored as a web-accessible URL or embedded in the response.
- Single outputs can be directly embedded in the response without any XML.

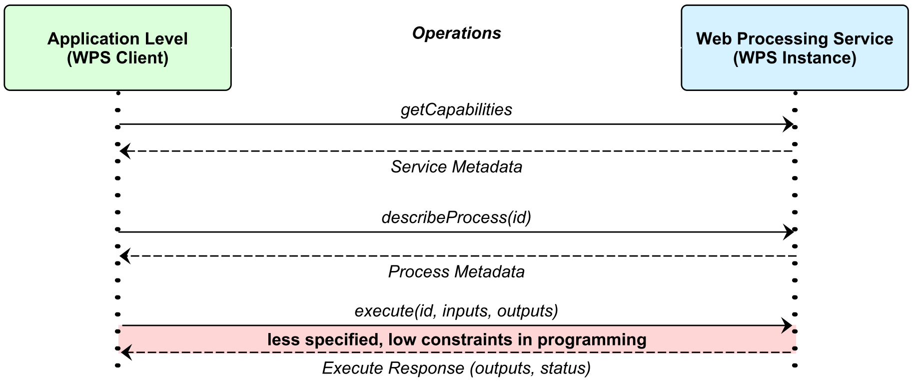

Figure 2.3: Mandatory operations of a WPS

- WPS supports multiple input and output formats.
- WPS supports synchronous requests, useful for fast calculations, and asynchronous requests, useful for long-running processes.
- WPS supports Simple Object Access Protocol (SOAP), a protocol standardized by the World Wide Web Consortium (W3C) for the exchange of data between applications.
- WPS supports Web Services Description Language (WSDL), a description language standardized by the W3C to describe the interfaces of web services.

Using WPS the complexity of data processing can be reduced by providing ready-to-use algorithms. This approach competes with traditional script-based methods, which lack a standardized interface. The concatenation of processes enables the mapping of entire workflows, which has always been a basic principle of GIS workflows. This makes highly complex processes, for example in meteorology or geophysics, easy to use and interoperable. By SOA the one-time provision of processing is sufficient, which can then be used from any point of the network. The maintenance of processes in a central place is simplified, since it only has to be carried out by the person who created it. It is possible to use the computing power of central high-performance computers. The advantages of WPS are manifold and there is a need for a standard in the age of digitization and automation. However, based on the available literature, geoprocessing is not yet very widespread on the web (Hofer, 2015). Success and widespread use depend, among other things, on specific applications for the general public. The use of WPS in the context of the increasing focus on digitization within municipal administrations can be an advantage for this technology, which would benefit from itself as it becomes more widespread.

CHAPTER 3

Freiburg's spatial data infrastructure

Freiburg maintains a municipal SDI and is thus at the lower, local level in the hierarchy of territorial authorities. As described in section 2.2, the number of different topics and the complexity of the underlying data models increase at this level. This increases the challenge to meet the general requirements of an SDI, such as interoperability, as well as the municipal responsibilities.

3.1 Responsibilities

The legal obligation of a local SDI to provide geodata is defined at the European level by INSPIRE, whereby the municipal involvement differs greatly from the involvement of regional or national SDIs. The compulsory provision of spatial data by the SDI Freiburg mainly comprises the area of urban land-use planning, for example local plans, land use plans and redevelopment areas (Königer et al., 2017). The implementation of INSPIRE leads to further laws at national level, such as the Information Re-use Act (IWG) for the re-use of information of public authorities, the Geodata Access Act (GeoZG) for the access to digital geodata, or the E-Government Act (EGovG) for the promotion of electronic administration. Many of these laws again lead to country-specific laws, which then have to be implemented at the municipal level and to which also the SDI Freiburg has to adhere.

In addition to laws, there are also resolutions that must be implemented within a certain period of time. A current example is the introduction of the XML based exchange standards XPlanung and XBau by the end of 2022 for a higher semantic interoperability approved by the IT planning council. Freiburg's SDI is closely involved in the realization, because this topic is affected by INSPIRE. The aim is to improve data exchange and data use in construction and planning. This is the place where employees and citizens come into direct contact with the SDI, where they search for geodata in order to carry out spatial analyses, where they record new geodata, or where they have to retrieve information on a topic. It is the task of an SDI to make this contact as barrier-free and comprehensive as possible, with the aim to increase the quality of the geodata and the SDI itself. Legal requirements, technical decisions and requirements of employees and citizens define the framework and the capabilities of a local SDI. The following is an overview of the most important responsibilities beyond the basic administration of system components:

- Implementation of legal requirements and resolutions when affected.
- Compliance with laws, such as privacy or copyright, for example by using a user rights structure.
- Redundancy-free provision of spatial data for operational and planning-relevant processes within the city administration and for citizens as an assistance, such as primary school districts.
- Transformation of spatial processes with regard to higher interoperability to other (non-spatial) processes and the connection to the SDI.
- Transformation of manual and proprietary processes into an automatable and networkable structure regarding the digitalization of the city administration.
- Provision of adapted tools for the collection of geodata and the associated metadata, as well as the information about these data.
- Provision of interoperable geoservices according to an SOA.
- Supporting non-expert departments in solving spatial problems with regard to the objectives of an SDI and digitization.

This spectrum of tasks shows that a local SDI also plays a mediating role and how important a high degree of flexibility is. The technical architecture of Freiburg's SDI is primarily responsible for achieving a high degree of flexibility.

3.2 Specifications

The technical architecture of the SDI Freiburg specifies the scope of its possibilities. The foundation was laid in 2008 with a first WebGIS based on Mapbender, a Content Management System (CMS) for map applications and geodata services. With this decision also the use of FOSS for the SDI was determined, with which it remained until today.

The reason for it lies in the adaptability, flexibility and independence in the choice of the individual components as well as the priority on a high interoperability. This decision is based on the assumption that a municipal administration with its heterogeneous IT structure can best be supported by a flexible SDI. The SDI Freiburg consists of the following components:

Table 3.1: Technical components of the SDI Freiburg

Component	Description
Hardware	Fully virtualized system with VMware Workstation
Operating system	Suse Linux Enterprise Server (SLES)
Database management system	PostgreSQL with PostGIS as spatial extension
Map creation system	UMN MapServer for WMS, WFS, WCS
WebGIS client	Mapbender for geodata presentation and Leaflet for small maps
Desktop GIS	QGIS as widely used Geographic Information System (GIS)
Metadata information system	GeoNetwork for collection, distribution and harvesting of metadata and for the CSW

The entire architecture of the SDI Freiburg today relies largely on FOSS. Due to the cooperation between OSGeo and OGC there are good technical prerequisites for a high syntactic interoperability. For special requirements there are further applications, such as ArcGIS or AutoCAD Map, but these are insufficiently connected to the SDI due to a lack of compatibility. As an extended intermediate level there is a user rights management to fulfill the requirements of privacy. This and other essential components are shown in fig. 3.1. A difficulty is the connection between Intranet and Internet, because this is very restrictive and currently no direct connection between SDI components of the Intranet and the Demilitarized Zone (DMZ) accessible from the Internet. Therefore it is necessary to run the essential components like MapServer and database twice, in the Intranet and Internet.

The actual interoperability depends on how consistently the possibilities of the components mentioned have been implemented in the sense of an SOA. If all geodata are only published as WMS, their further use for evaluations is more restricted than with intensive use of WFS. If no metadata is provided in a standardized and searchable way, data and services cannot be found. The SDI Freiburg has published more than 200 WMS since 2008, but only about 20 WFS. For the most part, analyses are done directly on the database. Especially thematic geodata only show a small share of semantic interoperability in the database. With regard to the use of WPS this detail can turn out to be obstructive.

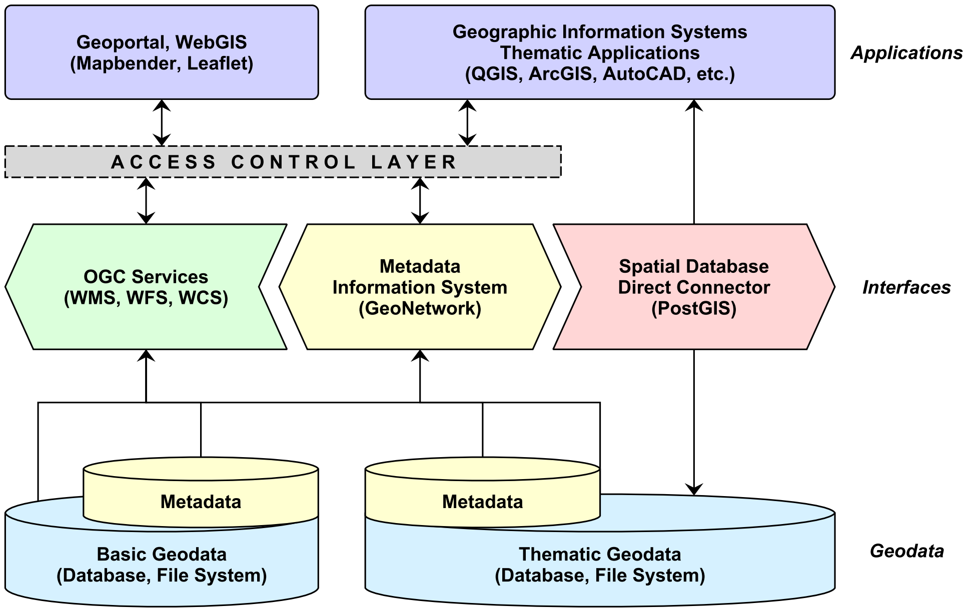

Figure 3.1: Schematic structure of Freiburg's SDI

To create a WPS process an implementation of the WPS standard is required. The choice of an implementation depends on many factors. Apart from the desired version of the implemented WPS standard and the supported features, the technical environment, in which the WPS is to be executed, is of central importance. For example, in Freiburg Java is rarely used (GeoNetwork) and is avoided because of its limited scripting capabilities and high memory consumption. Python, on the other hand, is often used because of its simplicity, extensibility and scripting possibilities. Another important component are Python implementations of program libraries like the Geospatial Data Abstraction Library (GDAL), the OGR Simple Features Library (OGR) and PROJ for the conversion of map projections. The components available on the server limit the selection of possible WPS implementations (table 3.2). Due to the properties of Python and the available knowledge the choice falls on PyWPS as WPS implementation. The PyWPS support for the WPS standard currently only applies to v1.0.0, but support for v2.0.0 is under development. Future features like transactional WPS are also planned for the release of PyWPS v4.4.0. PyWPS is one of the first implementations of the WPS standard and is officially funded as an OSGeo project. It remains to mention that as client QGIS is used with WPS client plugin. The submitting of XML requests as HTTP *POST* is done with the Firefox add-on *RESTClient*, a debugger for Representational State Transfer (REST) web services.

Table 3.2: Selection of WPS implementations

Implementation	Description
WPSint	Open source Java implementation for WPS v0.4.0
deegree	Open source Java implementation for WPS v0.4.0 and v1.0.0
GeoServer WPS plugin	Open source Java implementation for WPS v1.0.0
PyWPS	Open source Python implementation of WPS v1.0.0
WPS.NET	Open source .NET implementation of WPS v1.0.0
ZOO project	Open source C-Python-JavaScript implementation of WPS v1.0.0 and v2.0.0

This makes the technical environment of the SDI Freiburg complete. The findings from this master's thesis can only be transferred to technically similar architectures. For a proprietary SDI the findings would not be directly transferable, because the possible approaches would be too different. The staffing of three persons from the field of geosciences should also be mentioned, as this has an influence on the administrative capacities.

3.3 Common questions and solution approaches

When investigating the question of whether WPS can provide real added value for a local SDI, spatial questions from the everyday life of a municipal administration must first be considered as well as their previous approaches to solutions. Many spatial questions come from the field of urban planning, but spatial questions also accumulate in building law and in the social and citizen-oriented departments. Often, subject-specific applications are integrated that come into contact with geodata before or after processing, but do not have any interfaces for it. The following is a selection of everyday problems sorted by their frequency and starting with the largest:

1. Data delivery: In urban planning, contracts with engineering firms require regular transfer of up-to-date data records for a specific area. Data records in file form are also often requested for external projects from industry and science. A spatial selection is often necessary beforehand. An independent handling by the persons concerned is not possible due to lack of knowledge or missing authorizations on database or file system.

    - Current solution: Manual handling by the SDI team.

2. Intersection: The Consulting Centre for Building and Energy Freiburg (BZBE) needs daily up-to-date intersections of addresses and parcels with almost all geodata provided in the SDI.
    - Current solution: Proprietary PHP script with ready-made SQL queries and simple front-end. Application by the responsible person, adaptation and maintenance by the team of SDI.
3. Geocoding: Triggered by the digital strategy of the city of Freiburg, requests for the conversion of data sets with indirect spatial reference into data sets with direct spatial reference are increasing. An example is the maintenance of daycare facilities mentioned in section 2.1.
    - Current solution: Proprietary Python script that simplifies the use of the BKG geocoding service for mass processing with municipal CSV files. Automated on the SLES operating system, control of the results by the responsible person.
4. Reverse geocoding: For various purposes, address lists for a specific planning area are required from time to time. Often these requests come from offices without any reference to spatial data or GIS.
    - Current solution: The team of SDI linked the identification numbers of the affected buildings with the address database as an SQL query on the database.
5. Buffer, union and other operations: Representative also for other spatial operations, which often have to be applied by non-technical offices according to a certain rule. An example is the topic of building radio systems of the ABK. On the basis of an address, the corresponding building geometry must be buffered according to a certain formula which represents the range of the radio system.
    - Current solution: Trigger on the geodatabase, data acquisition with QGIS by the responsible person.
6. Evacuation radii: For different purposes it is necessary to derive an evacuation radius based on different parameters. Examples are planning of training missions or actual police or fire brigade missions, for example during floods or an EOD case.
    - Current solution: Manual drawing on a printed map or manual analysis with QGIS by the SDI team.

These examples illustrate the diversity and scope of the responsible tasks of a city administration compared to a specialist authority such as the State Institute for the Environment Baden-Württemberg (LUBW). Are WPS processes now the only solution? No, because there are already alternative solutions for all the questions mentioned. However, these are not interoperable, often time-consuming or have unacceptable qualitative shortcomings. The case study will show whether the use of WPS processes can better answer some of these questions. This requires the definition of applicability criteria with which the benefit of such processes can be empirically measured.

3.4 Applicability criteria for WPS

In order to investigate whether the development of processes based on WPS has advantages over a proprietary solution using scripting, suitable criteria must be defined in relation to a local SDI. These criteria must reflect the manageability on the part of the administrators, the technical capabilities of the implementation, as well as the user-friendliness on the part of the users. These characteristics are covered by the terms reusability, compatibility and usability.

3.4.1 Reusability

The development of WPS processes can be very complex. If the process flow required by the user can be implemented equally with a GIS, for example QGIS (graphical modeler), the question of the further benefit, and thus the reusability, arises. So that the effort of the development is worthwhile the processes must be reusable for other questions, either for single use or in a new process chain. In order to achieve this, a certain degree of atomicity or compactness must be taken into account during implementation. If the processes are not atomic or compact enough, the reusability can decrease. If, on the other hand, they are compressed too much, their number increases and with it the effort for development and maintenance. The following criteria are defined for the evaluation of reusability:

- Do at least two of the processes developed for the case study have a higher general potential for reuse?
- Is at least one of the processes developed for the case study practically reusable for one of the questions mentioned in section 3.3?
- Is it possible to use the available processes to create another process chain of at least two processes to answer a question?

3.4.2 Compatibility

As presented in section 2.3, interoperability is important for the communication and exchange of data between independent components. Moreover, the heterogeneous IT structure of a city administration in general and its SDI in particular places high demands on this property. Therefore it has to be examined, for example, whether WPS processes are adaptable enough, so that they can also be integrated into procedures outside the GIScience. How high the degree of interoperability and how flexible the adaptability of WPS processes is will be examined by means of the criterion of compatibility:

- Can the existing components of the SDI be used by a WPS with added value?
- Is the adaptability of a WPS sufficient to support the heterogeneous IT structure of a city, such as by integrating previously unintegratable technical procedures?
- Can the functionality of a WPS capable SDI be extended by externally provided processes?
- Does a WPS have any other side effects in terms of compatibility?

3.4.3 Usability

An SDI with all its advantages in a heterogeneous IT landscape, like the city of Freiburg, rises and falls with its usability for tools and geodata. Section 2.4 states that the dissemination of a technology or a standard also depends on concrete use cases. The acceptance required for this is not only necessary on the part of the users, but also on the part of the system operators. For the investigation of the applicability of WPS the criterion of usability is of crucial importance. The usability can be divided into the technical usability of the system operators to the actual WPS implementation, as well as the usability of the users to the WPS processes developed by the system operator. The technical usability includes the handling and the possibilities of the WPS implementation:

- Effort of integrating a WPS.
- Effort of adjusting and maintaining a WPS.
- Additional effort for the chaining of processes.
- Possibilities of simplification for the users.

When evaluating the technical usability that developers of WPS processes need in the context of a typical city administration, the following two aspects have to be considered:

1. Low maintenance: Because a local SDI like the one in Freiburg often has to be managed by only two to five people. In contrast, there is a high number of staff, ranging from 3000 (Ulm), 4000 (Freiburg) to 10000 (Leipzig), from whom more and more are involved in spatial issues.
2. High adaptability of the processes: Because a city administration has a very broad spectrum of tasks and therefore a very heterogeneous IT structure (section 3.2).

The usability, which concerns users in handling WPS processes, comprises the specific use case, with which effort and in which quality a question can be answered:

- Availability of the WPS.
- Need for clients and special software.
- Effort of answering a question.

When evaluating the usability that users of WPS processes place in the context of a typical city administration, the following two aspects have to be considered:

1. Available knowledge: The employees of a typical city administration come from a wide range of disciplines, but seldom have up-to-date IT knowledge in general or GIS knowledge in particular. Therefore, complexity must be hidden and GIS related or technical processes must be kept as simple as possible, especially if they are to be implemented in non-technical departments.
2. Changing responsibilities: The functionality provided should be available independently of individual computers and hardware-bound software licenses, so that modern workplace concepts such as desk sharing or home office are not an obstacle.

CHAPTER 4

Case study

4.1 Initial situation

To investigate the applicability of WPS for Freiburg's SDI, the criteria defined in section 3.4 must be applied to a case study. Such a case study should on the one hand cover a realistic use case and on the other hand cover as many facets as possible in order to be able to derive meaningful statements from it. Taking into account the properties of the WPS specification mentioned in section 2.4, a use case with the following peculiarities is sought:

- Connection of a spatial question with a non-spatial component.
- Origin of the question outside the GIScience.
- Answering of the question by personnel without GIS knowledge.
- Answering the question by involving several departments.
- Complex question for which a trained specialist from the field of GIScience is required up to now.
- Question for which there is so far no workflow in the sense of digitization and automation.
- A question that can be answered with a measurable improvement.

A use case with these peculiarities is often to be found in a municipal administration. The case studies described in section 3.3 also show these peculiarities to a large extent. Such questions can be answered by Python scripting, but without the advantages of a standardized interface and the integration of existing processes. For this reason an implementation of processes based on WPS is considered at all. Furthermore, a complex use case increases the probability of being able to reuse a part of the processes implemented for answering other questions and, together with the described properties, is an ideal candidate for investigating the applicability of WPS for Freiburg's SDI.

During a conversation with colleagues from the ABK, the team from the SDI Freiburg is in contact for the first time with the question of the determination of evacuation radii in the context of the disposal of explosive ordnance. And thus with a method developed at the Fraunhofer EMI for the physically highly precise derivation of such radii. The question of the integration of this method in Freiburg's SDI arose.

4.2 Explosive ordnance disposal

In Germany, dud bombs from the Second World War are still regularly discovered and must be removed. This task lies historically justified in the responsibility of the Federal States and is carried out by the Explosive Ordnance Disposal (EOD). A distinction is made between military and civilian EOD, and the latter is the subject of this case study. The civilian EOD has the task to protect public safety and order by removing objects and substances of military origin intended for warfare. In contrast to the military EOD, which primarily deals with tactical issues and damages are accepted, the priority of the civilian EOD lies in the avoidance of secondary damages by defusing.

The overall process of an EOD includes much more than just the part of defusing or controlled detonation. At the beginning there must be a suspicious case, which often occurs during construction work. This is reported to the local fire brigade or police department, and is followed by a direct report to the EOD service. The EOD service then begins with the historical exploration of the affected area. Archive material on combat operations, reports from earlier explosive ordnance finds and aerial photographs from the time of the Second World War will be evaluated. If a suspicious case is confirmed, an investigation with geophysical detectors is carried out on site and, as far as possible, the find is uncovered. From this point on, a comprehensive classification of the find takes place. Only after all parameters such as type, position, depth or TNT mass are known can the planning of the evacuation begin. Parallel the EOD service plans the defusing or, if necessary, the controlled detonation. Once the time has been set, the evacuation must be carried out under the command of the local police authority (AfO), so that only the minimum necessary risk must be taken. After successful disposal of the explosive ordnance, the evacuation order is rescinded and the EOD case is closed. As shown in the overall process (fig. 4.1) the responsibility of the municipality lies in reporting to the EOD service and especially in the planning and execution of the evacuation.

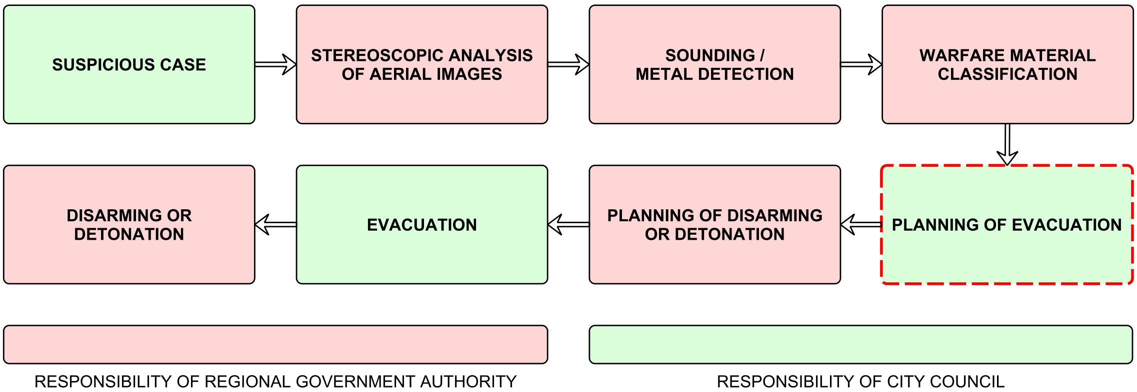

Figure 4.1: Responsibilities in an EOD case

The planning of an evacuation is a spatial question that can include a specialized component depending on the context. In this case the calculation of an exact hazard area for a certain selection of materials and substances. Within this danger zone, addresses, buildings and critical infrastructure must be identified in a short time so that those responsible can be involved as early as possible in the planning of the evacuation. In principle, the use of an SDI can have advantages in the determination of such geodata, as described in section 2.2. Whether the use of WPS processes according to the hypothesis will bring a significant improvement is now examined in a real application case.

4.2.1 Case definition

The scenario selected for the study is based on a real EOD case from March 2016[1]. 3500 people were affected during the evacuation. The details of the exact location and the results of the explosive ordnance classification performed by the EOD service were provided by the ABK for this master thesis (table 4.1).

The time available between the classification of the explosive ordnance and its defusing is from several hours to a few days, depending on the case. This is tight considering that several departments have to be involved. Within this period the evacuation radius must be defined and the planning and execution of the evacuation must be completed. The

1 `https://www.badische-zeitung.de/freiburg/fliegerbombe-im-stuehlinger-evakuierung-am-mittwoch--119843582.html` (visited on 10/09/2018)

Table 4.1: Properties of the EOD case from 2016

Property	Value
Date:	23.03.2016
Address:	Klarastraße 18, 79106 Freiburg i. Br., Baden-Württemberg, Germany
Coordinates:	LAT 47.99920° N, LON 7.84013° E
Location:	2.7 m below the earth's surface
Site:	found in a cave during construction work
Explosive ordnance:	247 kg aircraft bomb, unguided
Type:	MC multi-purpose bomb (standard version) of British origin
TNT:	110 kg of Composition B
Detonator:	mechanical with special design
Evacuation radius:	300 m fixed, with recesses

actual sequence of the steps under the responsibility of the city administration, with special regard to the geodata-related part, was as follows and is based on a conversation with the ABK and the AfO:

1. Report the explosive ordnance find to the EOD service.
2. Message to the departments concerned: ABK, AfO, Police Headquarters, Federal Police, Ambulance Service, Medical Service, Emergency Medical Service
3. Determination of the evacuation radius (fig. 4.2) after classification of the explosive ordnance:
   - Use of the municipal WebGIS for printing a raster map.
   - Estimation and drawing of an evacuation radius on the map.
   - Marking of recesses within the radius based on experience.
   - Manual colouring to distinguish between residential areas and public areas.
4. Search for critical infrastructure and involve those responsible. Affected: University Hospital Computer Centre, Black Forest Mountain Rescue Service, Central Station, Railway Signal Tower
5. Enquiry to the residents' register to identify the persons concerned.
6. Execution of the evacuation managed by the Integrated Control Centre of the ABK.

7. Securing the evacuation zone during the defusing of the explosive ordnance.

8. Orderly cancellation of the evacuation and archiving of the EOD case.

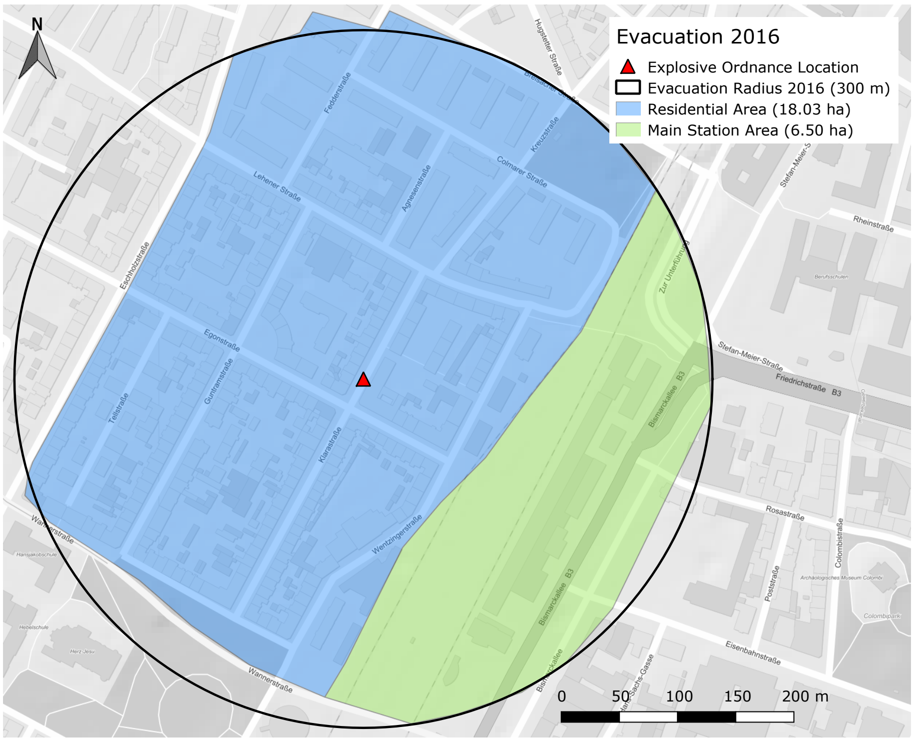

Figure 4.2: Evacuation map from 2016 used for the EOD case

The use of the digital infrastructure in general and the SDI in particular has so far been limited to the creation of a map (fig. 4.2) for orientation in evacuation planning. Requests, for example to the residents' register, address lists or affected buildings, are made manually between the offices. The structural sequence is always the same, only the content changes from case to case. A good prerequisite for the automation of processes in general. The data sources used partly lie outside the SDI, which can lead in the unfavorable case to the use of outdated data records.

4.2.2 Potential improvements

As the procedure described in the previous section shows, the geodata-related part is small. The accuracy of the evacuation radius can also be considered as volatile because it depends on the experience of the person in charge. A closer look at the entire evacuation planning process reveals two possible adjustment screws for potential improvement.

1. Integration of the APOLLO Blastsimulator (section 4.2.3) to improve the accuracy of the evacuation radius, the time required for it and the reduction of the dependence on a destruction estimation expert:
    - A higher accuracy of the evacuation zone gives more security in the affected area of buildings and public places. It can be assumed that human decisions based on experience are more conservative than purely numerical models. In case of doubt, a larger buffer than necessary is chosen, which is a considerable effort when evacuating hospitals or old people's homes.
    - Faster availability of the evacuation zone increases the time available for planning the evacuation, which is a great advantage especially in facilities with increasingly immobile people.
2. Use of SDI to facilitate access to the data sets needed and to shorten the time taken to make enquiries to other services.
    - Automatic selection of affected addresses, buildings and public spaces based on all resources available in the SDI.
    - Integration of the residents' register and the statistics database into the digital workflow.

The schematic representation in simplified form (fig. 4.3) describes the WPS as an interface between ABK, SDI and APOLLO Blastsimulator as an external component to answer important questions in evacuation planning.

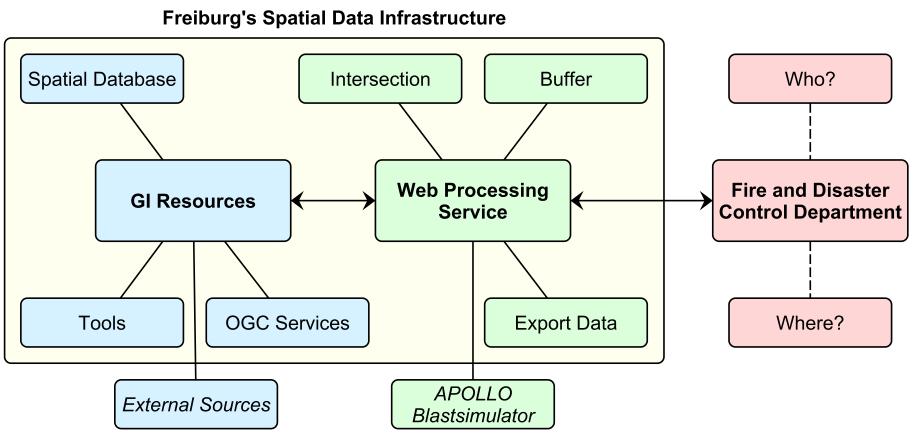

Figure 4.3: Integration of a WPS in an EOD case

4.2.3 APOLLO Blastsimulator

The discovery of unexploded ordnance can have a major impact on the infrastructure of a large city. The increasing densification of urban areas increases the need for precise information on the extent to which such areas are affected. Likewise, the growing corrosion of the fuse mechanisms within the bomb increases the risk of defusing it, so that controlled detonations must be used more frequently, as an example in Munich in 2012[1] shows. For this reason the Federal Ministry of Education and Research (BMBF) supports several projects[2] on civil security in the defusing of world war bombs. The three projects relevant for civilian EOD are:

1. DETORBA: The aim is to develop a method that simulates and analyses the effects of explosions in urban areas with unprecedented accuracy, thus enabling better planning of evacuation measures for bomb finds from the Second World War (Bettenworth, 2013). The project was completed in 2015 with a final report by Trometer (2015).
2. SIRIUS: The aim is to develop software for site-specific risk analysis for the deactivation of aircraft bombs. 3D city models in combination with physical methods will simulate the spreading of blast and splinter throwing. Special attention will be paid to an easy-to-use interface (Gebhard, 2018).
3. DEFLAG: The aim is to develop a procedure that minimizes the risks of a controlled detonation of explosive ordnance. With the help of a laser beam, the steel shell of the unexploded ordnance is to be notched and weakened so that there is not detonation but deflagration, which causes considerably less damage (Hermsdorf, 2016).

The APOLLO Blastsimulator is a Computational Fluid Dynamics (CFD) tool for the simulation of detonations, blast and gas dynamics, and is developed at the Fraunhofer EMI for High-Speed-Dynamics. With it it is possible to consider shading effects of buildings and thus to reduce the evacuation area to a smaller size than before. The calculation algorithms are based on the finite volume method with explicit time integration (Klomfass, Kirchner, et al., 2009), and the theoretical basis of explosions and their effects on the work of Kinney et al. (1985). A scientific review of the methods used in APOLLO was conducted by Klomfass, Stolz, et al. (2016).

---

1 https://www.dw.com/de/bombenentschaerfen-geht-das-auch-sicherer/a-43467568 (visited on 19/02/2019)

2 https://www.bmbf.de/de/blindgaenger-innovative-technik-zur-entschaerfung-4730.html (visited on 07/01/2019)

The software is part of the projects DETORBA and the current follow-up project SIRIUS, and supports the calculation of hazard areas. Past EOD operations have shown that it is desirable to specify the hazard zone as precisely as possible. For example, a radius of 500 m is often selected for air bombs of 250 kg, which is based on a rule of thumb of the EOD service $R = M$ [lbs] $\times$ 1 [m], whereas 2012 was only $350 - 500$ m when defusing a bomb of 1000 kg in Bochum (Trometer, 2015). The effects of bomb explosions are difficult to predict, especially in densely populated areas. First pilot experiments took place in the cities of Frankfurt am Main and Cologne. Important project partners from industry are CADFEM GmbH and virtualcitySYSTEMS GmbH.

APOLLO requires various input data and parameters for the explosion simulation, which are read in via a configuration file. The configuration is created via an interface in the form of a Java Servlet, which is to be completed in the second quarter of 2019 as part of the SIRIUS project. This interface converts the geodata, bomb parameters and location information entered by an expert into a valid configuration for APOLLO. This step serves the simplification, so that APOLLO is usable also by non-experts in the field of computer science and physics, which is likewise a goal of SIRIUS. The input parameters required for the interface and thus for the simulation are:

- 3D city model as CityGML and Digital Elevation Model (DEM) as GeoTIFF. The STL transformation is implemented as part of the Java Servlet.
- Exact location of the find spot in Cartesian coordinates in meters.
- Relative height of the bomb in meters.
- Exact TNT blast power in kilograms.
- Precision used by APOLLO simulation in meters.
- Position of the bomb as azimuth angle and tilt angle in degrees.
- Type of the bomb after classification, for example GP100 or GP250.
- Position of detonator after classification, like front, rear, top, bottom.
- Site description, for example surface or cavern, with size in meters.
- Destruction curve the evacuation zone will be calculated for, like float glass, hardened glass, safety glass, masonry, eardrum rupture, injury, lethal injury.

After all necessary data is available and the configuration file is generated, the calculation process starts. Depending on the choice of the desired precision, the number of objects from the city model and the available hardware, the process takes a few minutes to several hours. The STL file is internally converted to a voxel approximation and a local coordinate system is defined with the exact location of the find in the origin. The real time interval of the simulation is defined by the global maximum overpressure until it falls below a critical amplitude (fig. 4.4). In the course of the calculations the spatial distributions of the peak overpressure (fig. 4.5) and the maximum overpressure impulse are recorded. With these values specific destruction or injury characteristics are evaluated, for example float glass damage, eardrum rupture, masonry or lethal injury (fig. 4.6). All the characteristic curves are based on physical damage models and empirical values and help in operation planning, for example as a special hazard area for police officers with protective suits or as a death zone in which only the defusing experts are allowed to stay. For the calculation of the evacuation zone, the characteristic curve for float glass damage is to be used as a basis, for which a hazard to persons can be assumed.

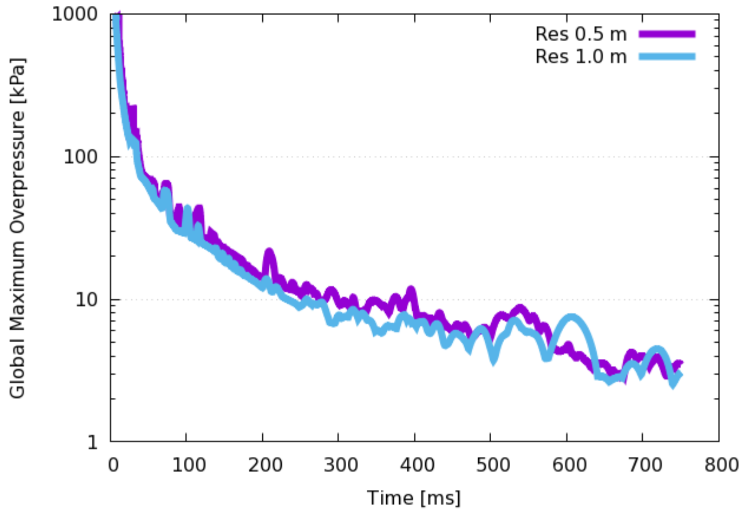

Figure 4.4: Overpressure falls below a critical amplitude at 0.75 s (Fraunhofer EMI)

The result of the simulation is stored in the binary Visualization Toolkit (VTK) format. In addition, APOLLO provides a text-based DAT file for a better understanding of the internal voxel grid structure, which is also used for processing by the WPS. The result file contains the values for peak overpressure and overpressure impulse per voxel as well as the estimate values for each considered characteristic curve. For the derivation of the evacuation area, the values per characteristic curve are relevant. These values estimate how high the risk of a voxel is for the selected damage characteristic curve (fig. 4.7):

- If a load condition is clearly above a characteristic curve, the location is colored red; there is a danger with great certainty.
- If a load condition is clearly below a characteristic curve, the location is colored blue; there is no danger with great certainty.
- If a load condition is close to the characteristic curve, the location is coloured grey and can be regarded as an evacuation edge.

The grey area is around the value of 0.50 and corresponds to 100 % of the damage characteristic curve. A value of 0.35 corresponds to 50 % and a value of 0.65 corresponds to 150 % of the damage characteristic curve. According to estimates of the Fraunhofer EMI and experts of the EOD, the value 0.50 is conservative and safe. The result of the explosion simulation must then be converted into a two-dimensional geometry by means of Python or another programming language, with which further spatial operations, such as selections or intersections, can be carried out. An example result can be found on GitLab[1].

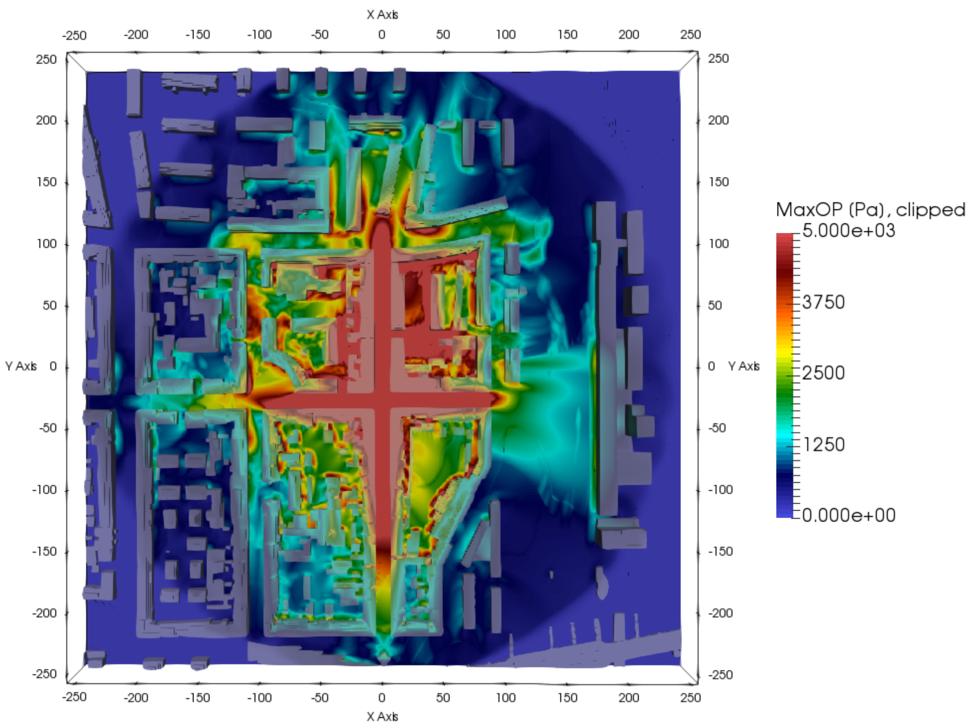

Figure 4.5: Distribution of overpressure amplitudes (Fraunhofer EMI)

1 https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/misc/apollo_effects.dat

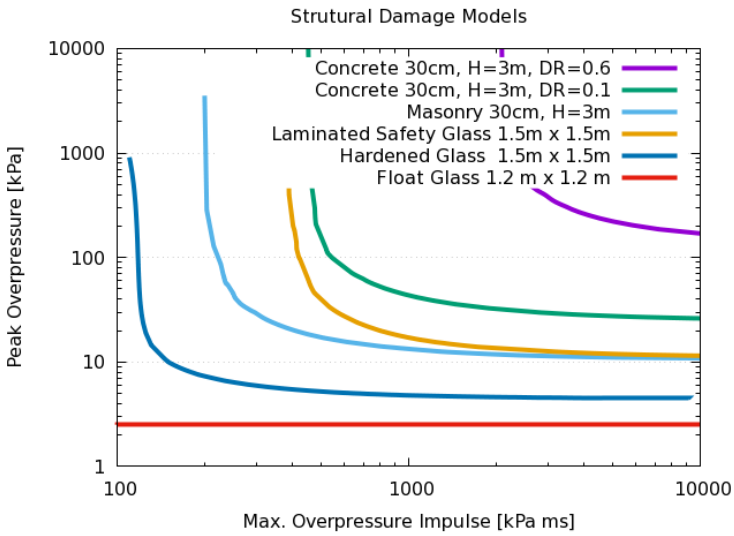

Figure 4.6: Characteristic curves based on physical damage models (Fraunhofer EMI)

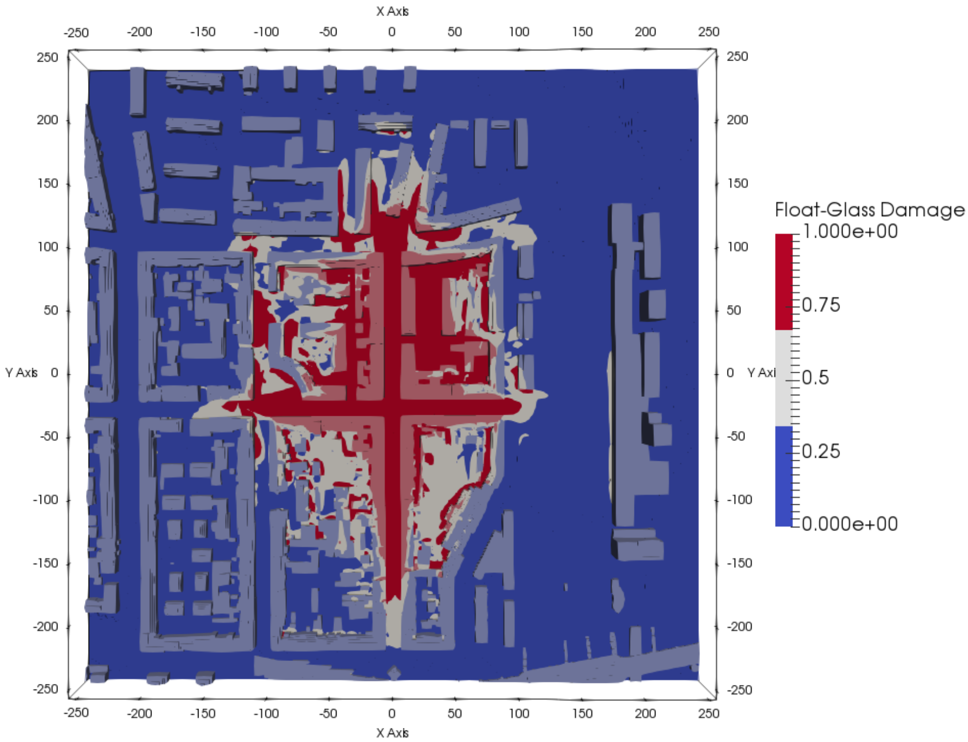

Figure 4.7: Distribution of float glass damage (Fraunhofer EMI)

4.3 Process identification

In order to implement the potential improvements (section 4.2.2), the geodata-related area of the entire process flow must be examined more closely and presented in a clearly understandable scheme. With the help of this schema, required sub-processes and their delimitations can be identified.

4.3.1 Schematic workflow

The way of thinking and working necessary for the implementation of the workflow was determined in discussions with the ABK and the AfO. It came out that the geodata-related part contains two temporally sequenced part workflows:

1. Quick preselection (fig. 4.8): Immediate identification of affected infrastructure for an early information policy. The rapid preselection includes a very large and secure radius, because it is still done before the exact classification of the explosive ordnance by the EOD service.

2. Accurate evacuation zone (fig. 4.9): Determination of the minimum evacuation line that must be drawn for a safe EOD, and thus the actually affected infrastructure. This part of the workflow can only take place after the explosive ordnance has been classified.

The quick preselection workflow starts as soon as an actual explosive ordnance find has been confirmed by the EOD service. Due to the longer duration of the classification and the complex calculation process of the APOLLO Blastsimulator, a quick preselection is necessary for an early information to important actors. The basis of the calculation is an approximate initial estimation of the TNT quantity.

The accurate evacuation zone workflow starts as soon as the classification and thus the actual hazard potential is known. The all-clear can be given for objects and infrastructure affected in the preselection, which are no longer affected after the calculation of the accurate evacuation zone. The two prepared schemata form the basis for the derivation of the individual WPS processes and their delimitations.

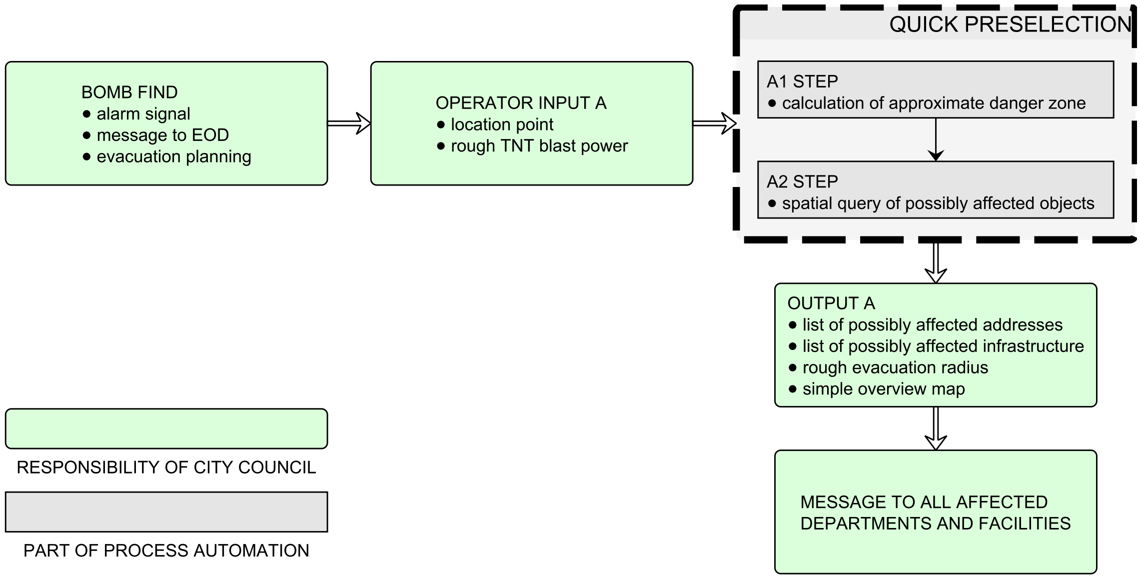

Figure 4.8: Flowchart for the quick preselection in an EOD case

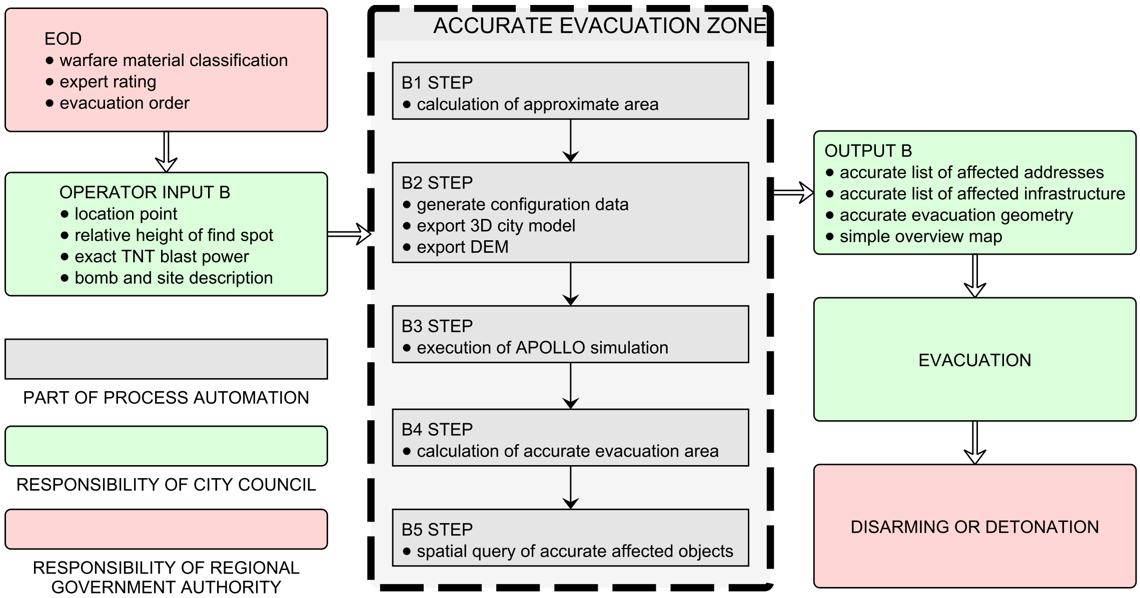

Figure 4.9: Flowchart for the accurate evacuation zone in an EOD case

4.3.2 Derivation of processes

The derivation of processes contains two challenges. On the one hand, the transfer of schematic processes to a process logic that meets all necessary requirements. On the other hand, the individual processes must be abstracted and delimited far enough so that they can also be reused for other questions. For this purpose, the problems frequently arising in a municipal SDI (section 3.3) must be kept in mind. Furthermore, the degree of abstraction of the processes must not increase arbitrarily, so that the development and administration effort does not exceed the human resources of a city administration. The goal of WPS processes in a local SDI is not the maximum atomicity but the best possible answer to common questions.

The two partial workflows show that the quick preselection flowchart is technically covered by the accurate evacuation zone flowchart. The steps are identical because $A1 = B1$ and $A2 = B5$. All processes derived from the accurate evacuation zone flowchart are described in table 4.2:

Table 4.2: List of processes required for an EOD case (* EOD only)

Step	Process	Function
B1*	Rough Distance	The process is needed for an initial estimation of the affected area and returns a danger distance based on TNT blast power.
B1	Buffer	The process is needed for an initial estimation of the affected area and returns a buffer around an input feature.
B2	Export 3D Data	The process returns 3D related spatial data for the APOLLO simulation, selected by an input geometry.
B2*	APOLLO Configuration	The process takes user input and returns APOLLO configuration data for the SIRIUS interface.
B3*	APOLLO Simulation	The process executes APOLLO via SIRIUS and returns a blast effects result.
B4*	Blast Effects Analysis	The process returns an accurate evacuation zone around the blast affected area.
B5	Export Affected Data	The process returns a subset of given or fixed spatial data selected by an input geometry.

As can be seen in table 4.2, generally applicable processes as well as processes only usable in the context of APOLLO or an EOD case could be identified. Likewise the steps $B1$ and $B2$ were split into two subprocesses and abstracted, because thereby a reuse for other communal problems becomes possible. The reason for several APOLLO processes is above all the greater flexibility in the chaining of the processes, for example for the less extensive quick preselection workflow or an additional blast effects analysis independent of APOLLO. The division of complex processes into several non-complex processes also provides a better overview and simplifies the implementation and administration of the entire component. The robustness also increases, because in the event of an error in communication between the Intranet and the Internet, or SDI Freiburg and Fraunhofer EMI, only a single sub-process is affected.

4.3.3 Definition of inputs and outputs

After deriving the individual processes from the schematic steps in the flowchart, the basic distribution of tasks for implementation as a process chain is defined. Before the implementation can begin, the necessary inputs and outputs must be clarified in detail. The table 4.3 contains only the inputs and outputs actually required for the accurate evacuation zone workflow. In the final implementation further optional inputs and outputs will be defined. The exchange of geodata is done by GML for vector data and GeoTIFF for raster data. Further details on inputs and outputs, such as data types or optional and mandatory parameters, are discussed in chapter 5.

Table 4.3: List of the minimum required inputs and outputs of the processes

I/O Name	Description
	Process: Rough Distance
In: TNT	Approximate initial estimation of the TNT quantity blast power.
Out: Distance	*Conservative hazard distance to explosive ordnance.*
	Process: Buffer
In: Geometry	GML geometry for which a buffer is to be created.
In: Buffer Size	Size of the buffer to be applied to the input geometry.
Out: Geometry	*Input geometry buffered by a certain size.*
	Process: Export 3D Data
In: Geometry	GML polygon geometry for spatial selection of 3D related data.
Out: DEM	*Selected section from the Digital Elevation Model (DEM).*
Out: 3D City Model	*Selected section from the 3D city model of Freiburg.*
	Process: APOLLO Configuration
In: Geometry	GML point geometry as exact location of the find.
In: TNT	Exact TNT blast power classified by the EOD service.
In: Precision	Accuracy of the calculation used by APOLLO simulation.
In: Height	Relative height of the bomb to consider shadowing effects.
Out: Configuration	*APOLLO configuration data in JSON format for SIRIUS interface.*
	Process: APOLLO Simulation
In: Configuration	APOLLO configuration data in JSON format for SIRIUS interface.
In: DEM	Selected section from the DEM.
In: 3D City Model	Selected section from the 3D city model of Freiburg.
Out: Blast Effects	*Voxel grid file with the values calculated by APOLLO.*
	Process: Blast Effects Analysis
In: Blast Effects	Voxel grid file with the values calculated by APOLLO.
In: Damage Level	Level of damage the evacuation zone will be calculated for.
Out: Evacuation Zone	*GML polygon geometry as evacuation zone around blast affected area.*
Out: Raster	*Blast affected area as non-aggregated georeferenced raster file.*
	Process: Export Affected Data
In: Geometry	GML polygon geometry for spatial selection of vector data.
Out: GML Data	*Selected subset of spatial data as GML file.*
Out: Geometry	*Same GML polygon geometry from input for verification.*

CHAPTER 5

Implementation

5.1 The PyWPS framework

As described in section 3.2, PyWPS is well suited for an environment like Freiburg's SDI and its stable version 4.0.0 is used for this thesis. As a server side implementation of the WPS standard in version 1.0.0 PyWPS is using the Web Server Gateway Interface (WSGI) calling convention for web servers to forward requests to frameworks written in Python. This section shows the implementation of WPS processes using an intersection process as an example. First the Apache web server must be configured for PyWPS to set the permissions of the required working folders and make the WSGI script accessible. This is done with a small configuration file (listing 5.1).

```
WSGIDaemonProcess pywps home=/srv/www/wps user=wwwrun group=www processes=2 threads=5
WSGIScriptAlias /pywps /srv/www/wps/pywps.wsgi process-group=pywps

<Directory /srv/www/wps>
    WSGIScriptReloading On
    WSGIProcessGroup pywps
    WSGIApplicationGroup %{GLOBAL}
    Require all granted
    Allow from all
</Directory>

Alias /wps/output /srv/www/wps/output

<Directory "/srv/www/wps/output">
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Listing 5.1: Apache web server configuration for PyWPS

The WSGI instance works like a wrapper around the PyWPS server and expects a list of processes and a configuration file (listing 5.2). The full source code of the WSGI script can be found in listing A.1.

```python
# libs
from pywps.app import Service
from processes.proc_vect_intersect import VectIntersect

processes = [VectIntersect()]

# for the process list on the home page
process_descriptor = {}
for process in processes:
    abstract = process.abstract
    identifier = process.identifier
    process_descriptor[identifier] = abstract

# Service accepts list of process instances and list of configuration files
application = Service(processes, ['/srv/www/wps/pywps.cfg'])
```

Listing 5.2: Principle of WSGI wrapper importing a vector intersection process

Now the WPS is callable in principle, but errors are reported because the intersection process doesn't exist yet. Python supports object-oriented programming, so each PyWPS process is defined as a new class that inherits the properties and methods of the *Process* class of the PyWPS package. For this and for the further functionality different packages have to be imported. Most of them come from PyWPS, OSGeo and for certain methods like logging, URL and XML handling or self-written functions. Each class responsible for a WPS process can be divided into two sections. The first section is the constructor method, which defines a list of inputs and outputs as well as basic options and metadata for the entire process. The second section is the handler method, which implements the actual functionality and returns the processed result, in this case the intersection of two geometries. The structure for all process classes is shown in listing 5.3.

```python
# class definition of the process
class ProcessName(Process):
    # constructor method for inputs, outputs, options and metadata
    def __init__(self):
        input_1 = ComplexInput(...)
        input_2 = LiteralInput(...)

        output_1 = ComplexOutput(...)
        output_2 = LiteralOutput(...)

        inputs = [input_1, input_2]
        outputs = [output_1, output_2]

        # function for delegating method calls to a parent or sibling class
        super(ProcessName, self).__init__(...)

    # handler method obtains request object and response object
    def _handler(self, request, response):
        # read or parse input data

        # process data

        # write output data

        return response
```

Listing 5.3: Basic structure for all process classes

The class *VectIntersect* starts with the constructor method (listing 5.4), in which the inputs and outputs are defined and general properties like metadata via a *super* function are set. The intersection process needs two geometries to be intersected. Two inputs of the type *ComplexInput* are required, because they are complex data types and not simple alphanumeric characters. These are provided with an identifier so that they can be addressed via an XML request. The format of the data is also determined, in this case GML, which can be validated by an XML Schema Definition (XSD). The *super* function assigns an identifier, inputs and outputs, and various metadata to the process. At this point the support for storing data ($store\_supported = True$) and asynchronous mode ($status\_supported = True$) is also set.

```python
    def __init__(self):
        in_geom_a = ComplexInput(
            'in_geom_a',
            'Input Geometry A [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            mode=MODE.NONE
        )

        in_geom_b = ComplexInput(
            'in_geom_b',
            'Input Geometry B [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            mode=MODE.NONE
        )

        out_intersect = ComplexOutput(
            'out_intersect',
            'Intersected Geometry',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      encoding='UTF-8', validate=None)]
        )

        inputs = [in_geom_a, in_geom_b]
        outputs = [out_intersect]

        super(VectIntersect, self).__init__(
            self._handler,
            identifier='vect_intersect',
            version='1.0',
            title='Vector Intersection Process',
            abstract='The process returns intersected area of each input feature.',
            metadata=[Metadata('The process returns intersected area of each input feature.',
                               'http://geodev:8080/geonetwork/srv/ger/catalog.search?service=CSW&version=2.0.2'
                               '&request=GetRecordById&id=c850b578-8561-42fb-88d1-1ac9e3314cf4#/metadata/'
                               'c850b578-8561-42fb-88d1-1ac9e3314cf4')],
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )
```

Listing 5.4: Constructor method of the *VectIntersect* class of the intersection process

The handler method provides the actual functionality of a process and returns the processed result. First the data passed to the WPS must be read. PyWPS provides suitable methods for this, but these do not support the import of data within a process chain. For this functionality the entire status response XML of a request must be parsed, which is universally feasible with the extended response parsing library written for PyWPS in the context of this thesis (listing A.11). The parsing library is used for almost all read operations of the implemented processes. Their use is shown using the example of the intersection process in listing 5.5.

```
        # check if data is given by reference
        if request.inputs['in_geom_a'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_geom_a'][0].method == 'GET':
                # obtain input with identifier as file name
                in_geom_a = request.inputs['in_geom_a'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_geom_a'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_geom_a'][0].data

                # get content of LiteralData, Reference or ComplexData
                ref_url = varlib.get_output(etree.fromstring(in_response))

                # get GML file as reference
                r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
                data = r.content

                # create file, w: write in text mode
                filename = tempfile.mkstemp(prefix='geom_a_', suffix='.gml')[1]
                with open(filename, 'w') as fp:
                    fp.write(data)
                    fp.close()

                in_geom_a = filename
        else:
            # obtain input with identifier as file name
            in_geom_a = request.inputs['in_geom_a'][0].file
```

Listing 5.5: Read of input A within the handler method of the *VectIntersect* class

After both GML geometries are read in, they are internally transferred with the OGR Simple Features Library (OGR) into an OGR layer structure in which further processing takes place (listing 5.6). With the help of this library the spatial reference of geometry A is read and passed to the output layer. The output is declared as an empty layer in GML format. For a better handling of the intersection operation all single geometries of a layer are transferred into a geometry collection.

```
        # open file and layer of input a
        in_src_a = ogr.Open(in_geom_a)
        in_lyr_a = in_src_a.GetLayer()
        lyr_name_a = in_lyr_a.GetName()

        # open file and layer of input b
        in_src_b = ogr.Open(in_geom_b)
        in_lyr_b = in_src_b.GetLayer()
        lyr_name_b = in_lyr_b.GetName()
```

```python

        # get and set output spatial reference
        epsg = int(in_lyr_a.GetSpatialRef().GetAttrValue('AUTHORITY', 1))
        sref = osr.SpatialReference()
        sref.ImportFromEPSG(epsg)

        # create output file
        driver = ogr.GetDriverByName('GML')
        out_src = driver.CreateDataSource(lyr_name_a)
        out_lyr = out_src.CreateLayer(lyr_name_a+'_'+lyr_name_b, sref, ogr.wkbGeometryCollection)

        # create geometry collection of input a
        collect_a = ogr.Geometry(ogr.wkbGeometryCollection)
        for feat in in_lyr_a:
            collect_a.AddGeometry(feat.GetGeometryRef())

        # create geometry collection of input b
        collect_b = ogr.Geometry(ogr.wkbGeometryCollection)
        for feat in in_lyr_b:
            collect_b.AddGeometry(feat.GetGeometryRef())
```

Listing 5.6: Internal data handling within the *VectIntersect* class using the OGR library

In the last step both geometry collections are intersected, which is done with the *Intersection* method. The result of the intersection is returned to a new geometry collection and passed to the previously declared output layer as a new feature. Finally, the result is assigned to the inherited process response variable (listing 5.7).

```python
        # calculate intersection
        intersect_geom = collect_a.Intersection(collect_b)

        # create output feature to the file
        out_feat = ogr.Feature(feature_def=out_lyr.GetLayerDefn())
        out_feat.SetGeometry(intersect_geom)
        out_lyr.CreateFeature(out_feat)

        # free and reassign
        out_feat = None
        out_src = None

        # set output format and file name
        response.outputs['out_intersect'].output_format = Format(mime_type='text/xml', extension='.gml',
                                                          schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                                          encoding='UTF-8', validate=None)
        response.outputs['out_intersect'].file = lyr_name_a

        return response
```

Listing 5.7: Calculation of the response within the *VectIntersect* class

The whole source code with comments of the vector intersection process can be found in listing A.2. A valid execute request (listing A.12) with the complete XML response (listing A.13) can be also found in the appendix. With a WPS client like QGIS this process can be operated easily and user-friendly. Because QGIS recognizes the GML format required for the input geometries, other formats, such as Shapefile, can also be used, which are internally converted to GML before being passed to the process. The example shows the intersection of parts of Klarastraße and Egonstraße at the crossroads (fig. 5.1).

Figure 5.1: Using the intersection process as WPS with two Shapefiles in QGIS

The WPS also responds to the remaining two operations, *GetCapabilities* and *DescribeProcess*. In contrast to the *Execute* operation, these are preferred as Key-Value-Pair (KVP) with the HTTP *GET* method. Due to the complexity of a chained *Execute* operation, only the HTTP *POST* method is used. It might be looking more complicated to use XML over KVP, for a complex request it is more safe and efficient to use XML encoding. The KVP way for the *Execute* request can be tricky and lead to unpredictable errors. (Čepický, 2019)

> *GetCapabilities*: `https://geodev2/pywps?request=getcapabilities&service=wps&version=1.0.0`
> Response XML on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/wps_getcap_response.xml`

> *DescribeProcess*: `https://geodev2/pywps?request=describeprocess&service=wps&version=1.0.0&identifier=vect_intersect`
> Response XML on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/wps_describe_response.xml`

This is one of many ways to realize an intersection process with PyWPS, because it always depends on the actual case and the required features. The underlying OGR library also contains much more powerful methods for implementing spatial operations. The WPS standard does not set any conditions for the implementation of the processing itself. When it comes to inputs, outputs, the transfer of data do the requirements of the standard come into play.

5.2 Non-case-specific processes

This section gives a detailed overview (fig. 5.2) of all implemented processes, which cannot be assigned to a certain topic, like an EOD case, and describes their peculiarities. Also included are the two Python libraries *geolib* and *varlib* created during the implementation. All processes support asynchronous mode, are chainable, and allow optional outputs.

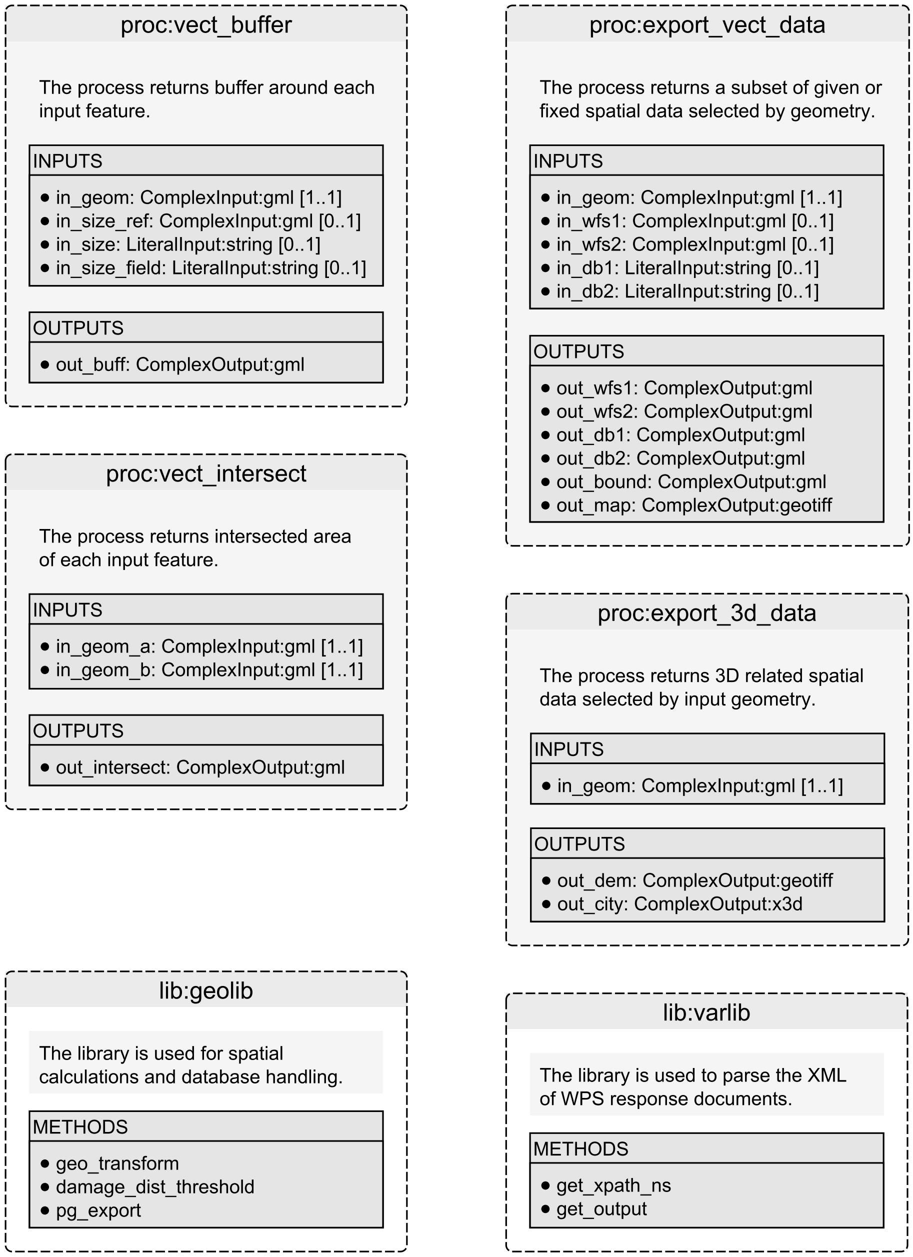

Figure 5.2: Overview of all non-case-specific processes and auxiliary libraries

Vector intersection process

The source code (listing A.2) has been explained in section 5.1. The execute request (listing A.12) and the XML response (listing A.13) can be found in the appendix.

Vector buffer process

The process returns a buffer around each input feature. The input GML may contain any number of geometries, but only the buffered geometries without attribute values are returned (listing 5.8). The value of the buffer size may be specified directly, referenced to a preceding process, or read from an attribute field of the input geometry. The output is a GML layer in the same reference system as the input layer. The whole source code with comments can be found in the appendix (listing A.3).

```python
        # make buffer for each feature
        while index < count:
            # get the geometry
            in_feat = in_lyr.GetNextFeature()
            in_geom = in_feat.GetGeometryRef()

            # check if size attribute exists
            if size_field in field_names:
                size_val = in_feat.GetField(size_field)
                if isinstance(size_val, int) or isinstance(size_val, float):
                    size = size_val
                else:
                    size = 0

            LOGGER.debug('Buffer Size:' + str(size))

            # make the buffer
            buff_geom = in_geom.Buffer(float(size))

            # create output feature to the file
            out_feat = ogr.Feature(feature_def=out_lyr.GetLayerDefn())
            out_feat.SetGeometry(buff_geom)
            out_lyr.CreateFeature(out_feat)

            # free and reassign
            out_feat = None

            index += 1
```

Listing 5.8: Buffer iteration over each input geometry within the *VectBuffer* class

> Request on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_vect_buffer.xml`
> Response on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_vect_buffer_response.xml`

Export vector data process

The process returns a subset of given or fixed spatial data selected by an input geometry. The choice of geodata from which to select is unlimited when using a WFS as input. Selections in the database, on the other hand, are permanently implemented and are selected per database slot from a topic list. Currently addresses, buildings, parcels, local plans and POI are supported, the list can be extended if necessary. The spatial selection is possible from up to four different data sources with one process call (WFS example in listing 5.9). The output consists of the selection geometry and the selected features including all attribute values in the GML Format. In addition, an overview map can be output as GeoTIFF. The selection geometry may exist in any reference system and is transformed to ETRS89 (EPSG: 25832) before the selection. All other input layers must already exist in this reference system and are also output in the same system. The whole source code with comments can be found in the appendix (listing A.4).

```python
        # check and obtain input with identifier as data directly
        if 'in_wfs1' in request.inputs:
            wfs1 = request.inputs['in_wfs1'][0].data

            # create file, w: write in text mode
            in_path = tempfile.mkstemp(prefix='wfs1_data_', suffix='.gml')[1]
            with open(in_path, 'w') as fp:
                fp.write(wfs1)
                fp.close()

            # open file and layer
            wfs1_src = ogr.Open(in_path)
            wfs1_lyr = wfs1_src.GetLayer()

            # get spatial reference
            wfs_epsg = int(wfs1_lyr.GetSpatialRef().GetAttrValue('AUTHORITY', 1))

            # check spatial reference
            if wfs_epsg == self.epsg:
                wfs1_lyr.SetSpatialFilter(geom)
            else:
                LOGGER.debug('Incompatible Spatial Reference of WFS1 and Selection Geometry.')

            # set output format definition
            out_path = tempfile.mkstemp(prefix='wfs_' + wfs1_lyr.GetName() + '_data_', suffix='.gml')[1]
            out_src = ogr.GetDriverByName("GML").CreateDataSource(out_path)
            out_src.CopyLayer(wfs1_lyr, wfs1_lyr.GetName())
```

Listing 5.9: Using *SetSpatialFilter* for selection within the *ExportVectData* class

> Request on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_export_vect_data.xml`
> Response on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_export_vect_data_response.xml`

Export 3D related spatial data process

The process returns 3D related spatial data selected by an input geometry. The choice of geodata is limited to the 3D city model in X3D format and a DEM as GeoTIFF, all in the reference system ETRS89 (EPSG: 25832). For the DEM a WCS is requested (listing 5.10), and for the city model an SQL query to a 3D City Database[1] has been made (listing 5.11). The whole source code with comments can be found in the appendix (listing A.5).

```
        if 'out_dem' in request.outputs.keys():
            # WCS request
            url = "http://mapbender/wcs7/verma_hoehen/verma_dgm?"
            wcs = WebCoverageService(url, version="1.0.0")

            # get a certain coverage
            dem = wcs['dgm1']

            # request parameters
            bbox = (bbx1, bby1, bbx2, bby2)
            crs = 'EPSG:' + str(self.epsg)
            file_type = 'GEOTIFF_16'  # GEOTIFF_16, AAIGRID, GTiff
            resx, resy = 1, 1  # max. available resolution of DEM data

            try:
                # get coverage request
                gc = wcs.getCoverage(identifier=dem.id, bbox=bbox, format=file_type, crs=crs, resx=resx, resy=resy)

                # create file, wb: write in binary mode
                dem_path = tempfile.mkstemp(prefix='dem_', suffix='.tif')[1]
                with open(dem_path, 'wb') as fp:
                    fp.write(gc.read())
                    fp.close()
            except owslib.util.ServiceException as se:
                dem_path = ''
                LOGGER.debug('WCS ServiceException:' + str(se))
```

Listing 5.10: Using a WCS for DEM selection within the *Export3dData* class

```
            # sql query with placeholders, transformation to local spatial reference
            query = sql.SQL("SELECT ST_AsX3D(ST_Transform(ST_SetSRID(sg.geometry, %s), %s), 3, 0) AS geom_3d "
                            "FROM {tbl} sg LEFT JOIN thematic_surface ts ON ts.lod2_multi_surface_id = sg.root_id "
                            "LEFT JOIN building b ON ts.building_id = b.building_root_id "
                            "WHERE sg.geometry IS NOT NULL AND ts.lod2_multi_surface_id IS NOT NULL "
                            "AND ST_Intersects(ST_SetSRID(ST_PolygonFromText(%s), %s), sg.geometry);")

            # execute command, using templating mechanism for better security
            db_cur.execute(query.format(tbl=sql.Identifier('surface_geometry')),
                           [self.epsg3, self.epsg, geom.ExportToWkt(), self.epsg3])

            # process query result data
            city_data = '<?xml version="1.0" encoding="UTF-8"?>\n' \
                        '<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.3//EN"\n' \
                        '  "http://www.web3d.org/specifications/x3d-3.3.dtd">\n\n' \
                        '<X3D profile="Interchange" version="3.3"\n' \
                        '     xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"\n' \
                        '     xsd:noNamespaceSchemaLocation="http://www.web3d.org/specifications/x3d-3.3.xsd">\n' \
                        '<Scene>'

            for city_geom in db_cur:
                city_data += '\n  <Shape>\n    ' + str(city_geom)[2:-3] + '\n  </Shape>'
```

1 https://www.3dcitydb.org (visited on 22/04/2019)

```

            city_data += '\n</Scene>\n</X3D>'

            # create file, w: write in text mode
            city_path = tempfile.mkstemp(prefix='city_', suffix='.x3d')[1]
            with open(city_path, 'w') as fp:
                fp.write(city_data)
                fp.close()
```

Listing 5.11: SQL query to the 3D City Database and creation of the X3D file

> Request on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_export_3d_data.xml`
> Response on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_export_3d_data_response.xml`

Supporting libraries

The support methods library `geolib` is used for methods like database handling or spatial reference transformations. Worth mentioning is the use of the *Psycopg* adapter for *PostgreSQL* and the templating mechanism to protect against SQL injection attacks. The XML parsing library `varlib` is used to parse the XML of WPS response documents and supports synchronous, asynchronous, single use and chained processes. The whole source code with comments can be found in the appendix (listing A.10 and listing A.11).

5.3 Case-specific processes

This section gives a detailed overview (fig. 5.3) of all implemented processes that can be assigned to the EOD topic and describes their particularities. All processes support asynchronous mode, are chainable, and allow optional outputs.

APOLLO rough danger distance process

The process is part of the EOD workflow and returns a rough danger distance based on a given solid and TNT mass. Both are defined as *LiteralInput* and of type *Integer*. The solid type is entered via a code list that currently accepts two types: float glass (0) and eardrum rupture (1). In the process chain, the value is set to float glass by the administrator to allow a sufficiently large preselection and to exclude critical operating errors from the user. Calculating the safe distance d is a very conservative approach to the real evacuation zone. For float glass damage the threshold value is at a peak overpressure of $f_1\left(\frac{52\,[\mathrm{m}]}{M^{1/3}}\right) = 3\,\mathrm{kPa}$,

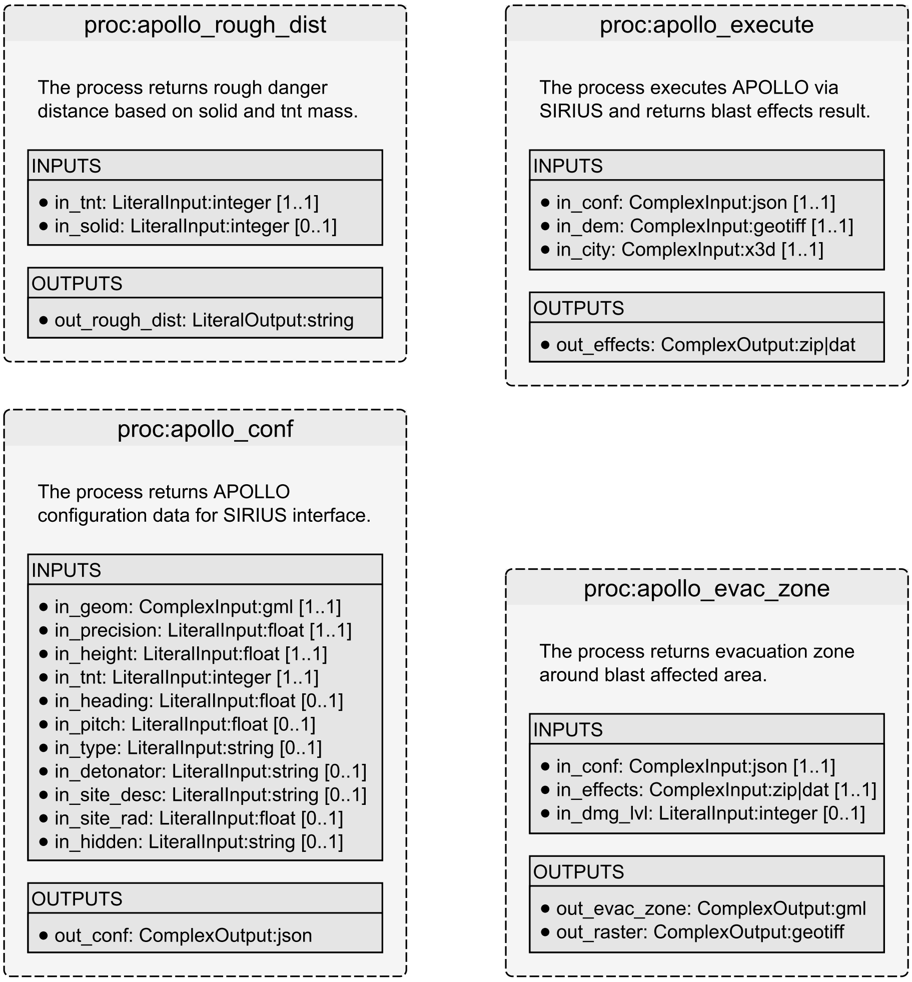

Figure 5.3: Overview of all case-specific processes

which means $d = 52$ m at a mass M of 1 kg TNT, or $d = 52\,\text{m} \times 1000\,\text{kg}^{1/3} = 520$ m at a mass M of 1000 kg TNT. For eardrum rupture the threshold value is $f_2\left(\frac{12.5\,[\text{m}]}{M^{1/3}}\right) = 17\,\text{kPa}$, which means $d = 12.5$ m for 1 kg TNT, or $d = 12.5\,\text{m} \times 300\,\text{kg}^{1/3} = 84$ m at a mass M of 300 kg TNT. These functions are based on curve fitting to experimental findings and were published by Kinney et al. (1985). The result is output as *RawDataOutput*. The whole source code with comments can be found in the appendix (listing A.6).

> Request on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_apollo_rough_dist.xml`
> Response on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_apollo_rough_dist_response.txt`

APOLLO configuration process

The process is part of the EOD workflow and returns APOLLO configuration data for the SIRIUS interface. The output is a JSON file (listing 5.12) generated from the inputs that is read by a Java Servlet so that the APOLLO Blastsimulator can be started with optimally adjusted parameters. The location is read as GML geometry, all other parameters are defined as *LiteralInput* and are based on different data types and code lists (fig. 5.3). They describe the location of the explosive ordnance and the explosive ordnance itself. Currently only the exact location, precision, relative height and exact TNT blast power are mandatory, all others are optional. A short description of the parameters can be found in the XML request and in the input definitions in the source code. The whole source code with comments can be found in the appendix (listing A.7).

```
        # create output data
        conf_data = EasyDict({'bomb': {'tnt': tnt, 'type': bomb_type, 'detonator': detonator},
                              'domain': {'name': 'Ultimo', 'zroi': 100, 'droi': dist_threshold},
                              'mode': {'name': 'Ultimo', 't': 50, 'precision': precision},
                              'site': {'type': site_desc, 'radius': site_rad},
                              'geometry': {'crs': self.epsg2, 'position': [x_wgs, y_wgs], 'depth': (-1) * height},
                              'crs': self.epsg,
                              'position': [x2, y2],
                              'height': height,
                              'heading': heading,
                              'pitch': pitch,
                              'extent': [bbx1, bby1, bbx2, bby2],
                              'hiddenObjects': hidden,
                              'service': {'url': self.srv_url, 'resultFile': 'effects_' + str(self.uuid) + '.zip'}
                              })

        # conversion to JSON format
        conf_json = json.dumps(conf_data)
```

Listing 5.12: Creation of the JSON file within the *ApolloConf* class

> Request on GitLab: https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_apollo_conf.xml
> Response on GitLab: https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_apollo_conf_response.xml

APOLLO execute process

The process is part of the EOD workflow, executes APOLLO via a Java Servlet developed as part of the SIRIUS project and returns a blast effects result. The three *ComplexInput* declarations consist of the JSON configuration file, the DEM and the 3D city model. As the result of the explosion simulation a blast effects file is output, which can then be analysed by the APOLLO evacuation zone process. The process can take several hours, so it must support asynchronous mode. Via a *while* loop a freely configurable URL is checked for its status code. Only when this status code is valid is the process continued. Until then *APOLLO Execute Process Still In Progress* will be output as XML status response (listing 5.13). The cancel operation *Dismiss* is only supported from WPS version 2.0. Currently a timer could limit the endless loop in case of an error. The whole source code with comments can be found in the appendix (listing A.8).

```
# open configuration file
with open(in_conf, 'r') as fp:
    conf_data = json.load(fp)

# read url for APOLLO service and result data
if 'service' in conf_data:
    srv_url = conf_data['service']['url']
    result_file = conf_data['service']['resultFile']
    srv_url_result = srv_url + result_file

# NON-PRODUCTIVE ONLY -> overwrite result data url because simulation of working SIRIUS / APOLLO server
srv_url_result = 'https://geodev2/apollo_result/apollo_effects.zip'

# reveal input data, execute APOLLO and calculate effects result
# r_exe = requests.get(srv_url, verify=False)

# effects result file checker
while not requests.head(srv_url_result, verify=False).status_code == requests.codes.ok:
    response.update_status('APOLLO Execute Process Still In Progress', 0)

# get effects result file when APOLLO is ready
r = requests.get(srv_url_result, verify=False)
data = r.content

# create file, wb: write in binary mode
result_file = tempfile.mkstemp(prefix='effects_', suffix='.zip')[1]
with open(result_file, 'wb') as fp:
    fp.write(data)
    fp.close()
```

Listing 5.13: Simulation of working SIRIUS interface within the *ApolloExecute* class

> Request on GitLab: https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_async_apollo_execute.xml
> Response on GitLab: https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_async_apollo_execute_response_status_finished.xml

APOLLO evacuation zone process

The process is part of the EOD workflow and returns an evacuation zone around a blast affected area. Required are two *ComplexInput* declarations consisting of the JSON configuration file and the blast effects file as a result of the explosion simulation. The configuration is used for the reverse transformation from the internal APOLLO coordinate system to ETRS89 (EPSG: 25832). In addition, another *LiteralInput* may be given, which defines via a code list for which level of damage the evacuation zone is calculated. Supported are all destruction curves relevant for an EOD case and considered by APOLLO: float glass (0, default value), hardened glass (1), safety glass (2), masonry (3), eardrum rupture (4), injury (5) and lethal injury (6). This can be used to output different evacuation zones, for example for the mentioned police officers with protective suits within the eardrum rupture area or the defusing experts within the death zone. The output consists of the evacuation zone as GML geometry and an evacuation grid in GeoTIFF format. The GML geometry is added with the attribute field `corr_buff` for a buffer value that corrects the pixel inaccuracy (section 5.4.2). The way there is a complex sequence of individual processing steps (fig. 5.4).

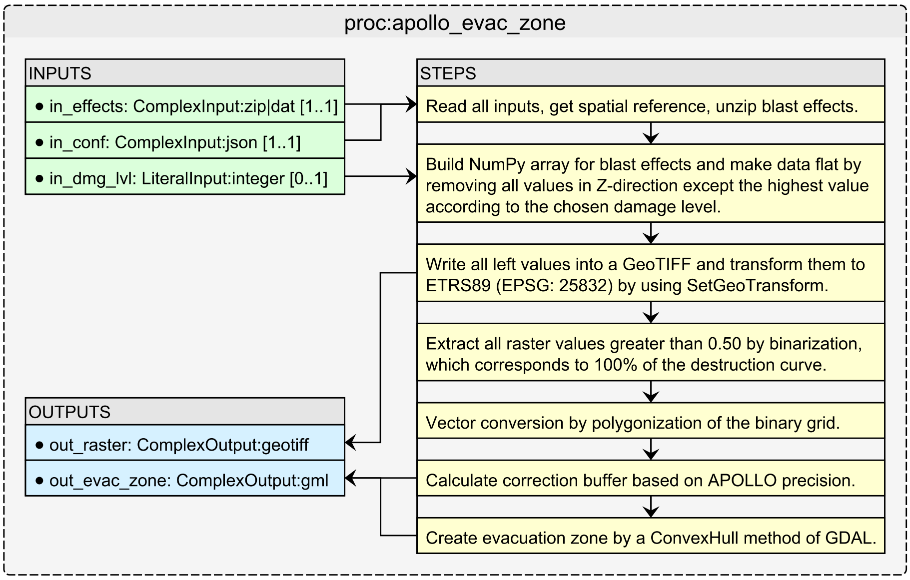

Figure 5.4: Overview of processing steps within the *ApolloEvacZone* class

The blast effects file can be a compressed zip file or uncompressed text file, the process supports both formats. Decisive is the read in as *NumPy* array, which converts the three-dimensional voxel grid structure into the two-dimensional plane (listing 5.14). The remaining values can then be transferred directly into a grid and stored as georeferenced TIFF (listing 5.15). The static class variable `rot_deg` is used for a counter-rotation, which was originally applied by APOLLO to the voxel grid file to increase the geometric approximation in the area of the find. In the future, APOLLO will manage this operation completely internally and can therefore be set to zero in this process. The resulting evacuation zone is based on a *ConvexHull* operation by OGR, which includes all polygon areas affected by the selected damage level. For these areas, the estimate of 0.50 introduced in section 4.2.3 applies. If necessary, a *LiteralInput* for free selection of this value could simply be added. The additionally output GeoTIFF contains all calculated estimate values from 0.0 to 1.0 and can be used by the expert to assess critical objects. The whole source code with comments can be found in the appendix (listing A.9).

```
        # build dtype array structure for APOLLO effects file
        dt = np.dtype({'names': ['I', 'J', 'K', 'Dir', 'N', 'Obj',
                                 'F1_MaxOP', 'F2_MaxOP-Imp', 'F3_OP-Imp', 'F4_FloatGl', 'F5_HardGl', 'F6_SafeGl',
                                 'F7_Masonry', 'F8_RC30-01', 'F9_RC30-06', 'F10_Eardrum', 'F11_Injury', 'F12_Lethal'],
                       'formats': ['int', 'int', 'int', 'int', 'int', 'int', 'float', 'float', 'float', 'float',
                                   'float', 'float', 'float', 'float', 'float', 'float', 'float', 'float']})

        # read APOLLO effects file
        data = np.loadtxt(in_effects_dat, skiprows=19, dtype=dt, ndmin=2)

        # get dimensions (I=512 J=512 K=76)
        size_i = np.amax(data['I']) - np.amin(data['I']) + 1
        size_j = np.amax(data['J']) - np.amin(data['J']) + 1

        # get delta of translation to positive quarter
        delta_i = abs(np.amin(data['I']))

        # max values, no abs, needed for iterations
        max_j = np.amax(data['J'])

        # empty array with size of ground surface
        target = np.zeros((size_j, size_i))

        # make data flat
        for row in np.nditer(data):
            # save value only if greater than previous value in K direction
            if row[dmg_lvl] > target[max_j - row['J']][delta_i + row['I']]:
                # save 1-dimensional value
                target[max_j - row['J']][delta_i + row['I']] = row[dmg_lvl]
```

Listing 5.14: Conversion of 3D voxel grid structure into a 2D plane

```
        # set spatial reference and export projection to wkt
        sref = osr.SpatialReference()
        sref.ImportFromEPSG(epsg)
        wkt_proj = sref.ExportToWkt()

        # number of pixels in x and y, and size of one pixel
        pixel_x = size_i
        pixel_y = size_j
        pixel_size = precision

        # transform location coordinates to upper left base point used in GTiff
        rot_rad = math.radians(-1 * self.rot_deg)
        size_i2 = size_i / 2.0
        size_j2 = size_j / 2.0
        delta_x = (size_i2 * precision) * math.cos(rot_rad) + (size_j2 * precision) * math.sin(rot_rad)
        delta_y = -(size_i2 * precision) * math.sin(rot_rad) + (size_j2 * precision) * math.cos(rot_rad)
        min_x = x - delta_x
        max_y = y + delta_y

        # set raster format definition
        raster = gdal.GetDriverByName('GTiff').Create(
            raster_path,  # file path
            pixel_x,  # width in pixels
            pixel_y,  # height in pixels
            1,  # number of bands
            gdal.GDT_Float32  # type of raster
        )

```

```
        # set transformation from pixel to projected coordinates
        raster.SetGeoTransform((
            min_x,  # x value at top left
            math.cos(rot_rad) * pixel_size,  # transform pixel size in west-east
            math.sin(rot_rad),  # rotation factor 1
            max_y,  # y value at top left
            math.sin(rot_rad),  # rotation factor 2
            -math.cos(rot_rad) * pixel_size  # transform pixel size in north-south
        ))

        # set projection for transformed coordinates
        raster.SetProjection(wkt_proj)

        # write simulated data to band 1
        raster.GetRasterBand(1).WriteArray(target)
```

Listing 5.15: Conversion of 2D *NumPy* array into a georeferenced TIFF

> Request on GitLab: https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_async_apollo_evac_zone.xml
> Response on GitLab: https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_async_apollo_evac_zone_response_status_finished.xml

5.4 Chaining of processes

After the implementation of all processes derived from the two workflows (section 4.3.1), they are available for concatenation. Thus, the entire procedure can be linked as one process chain and all required processing can be performed in one step. It should be noted that the vector intersection process is not used in the EOD workflow chain, but as an example process and for testing purposes. A distinction is made between user inputs, administrator inputs, process chain outputs and temporary or unused inputs and outputs (table 5.1). There is also an additional process chain output which can be useful for the user as an intermediate result of the APOLLO evacuation zone process: A not generalized GeoTIFF with single values from the explosion simulation. The problems concerning this and process chaining in general are discussed in section 5.5.4.

5.4.1 Quick preselection

The quick preselection chain realized with WPS matches to the developed schematic workflow (fig. 4.8). The corresponding asynchronous XML request, response status and response result can be found in the appendix (listing A.14, listing A.15 and listing A.16).

Table 5.1: Differences between all inputs and outputs of a process chain

Name	Description
Non-fixed user input:	Variable user input data from the user of the process chain.
Fixed administrator input:	Input data fixed by the administrator to simplify the handling and prevent user errors.
Process chain output:	Final data output for the user at the end of the process chain.
Additional process chain output:	Intermediate result data from a process within the chain that can be useful for the user.
Temporary or unused in / out:	All other input and output data generated or required by the process chain, without that user gets in touch with it.

5.4.2 Accurate evacuation zone

The accurate evacuation chain realized with WPS matches to the developed schematic workflow (fig. 4.9). Additionally a correction buffer between the APOLLO evacuation zone process and the export vector data process at the end of the chain was implemented. This corrects the pixel inaccuracy resulting from the selected precision for the APOLLO simulation. A precision p of 10 m produces a raster with a resolution of 10 m per pixel. The evacuation zone calculation is based on the center of a pixel, resulting in a correction buffer of $B = \sqrt{p^2 + p^2}$, which is calculated during the evacuation zone process and transmitted to the buffer process as an attribute value of the output geometry. The corresponding asynchronous XML request, response status and response result can be found in the appendix (listing A.17, listing A.18 and listing A.19).

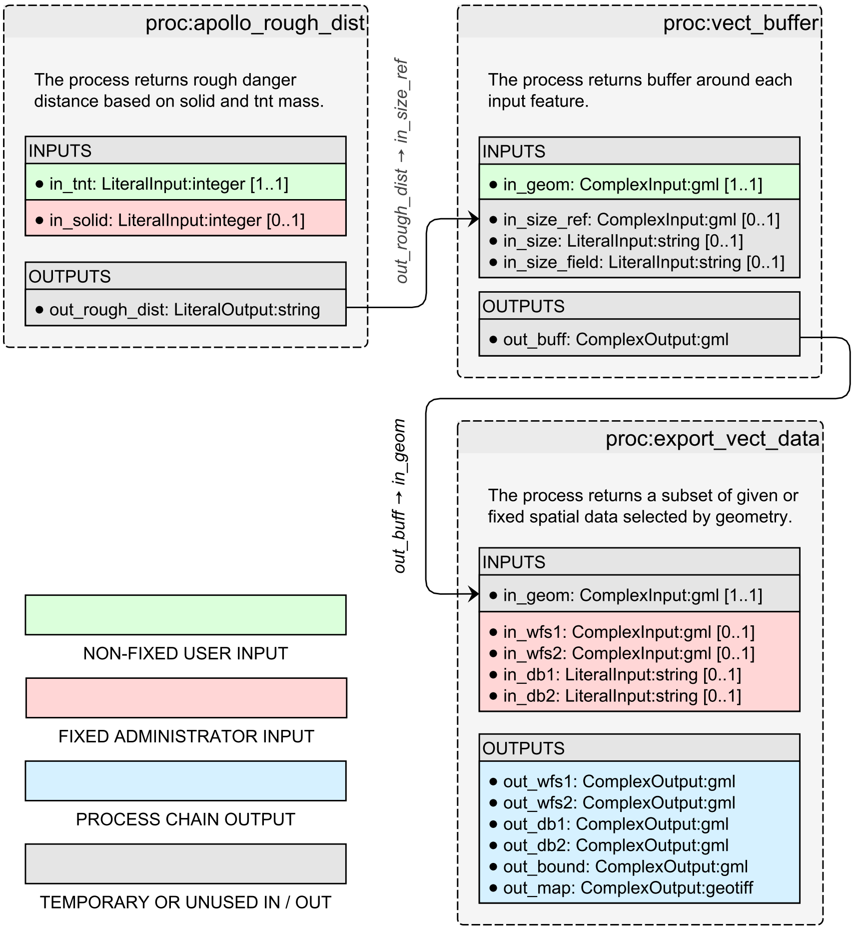

Figure 5.5: Overview of the quick preselection process chain

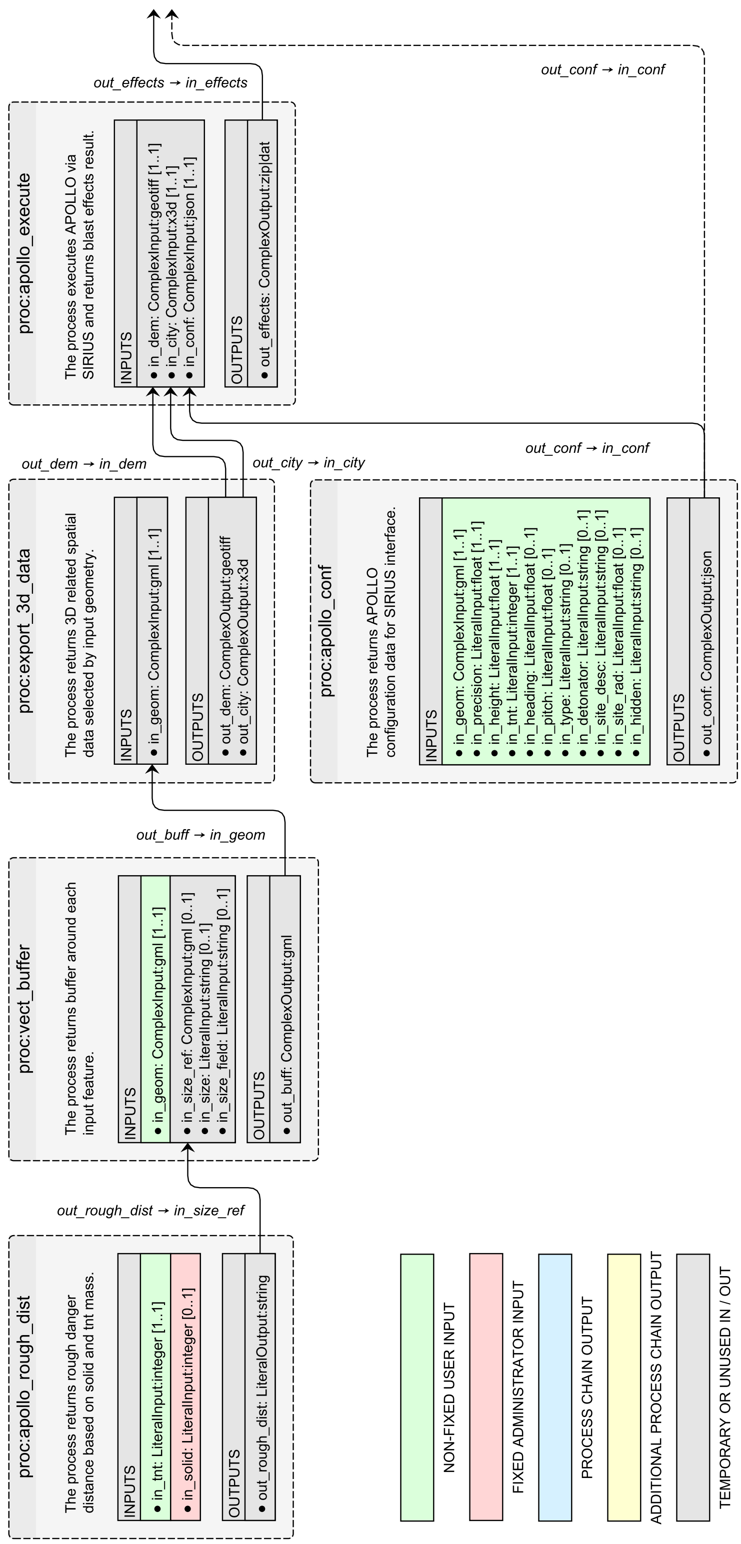

Figure 5.6: Overview of the accurate evacuation zone process chain, part 1

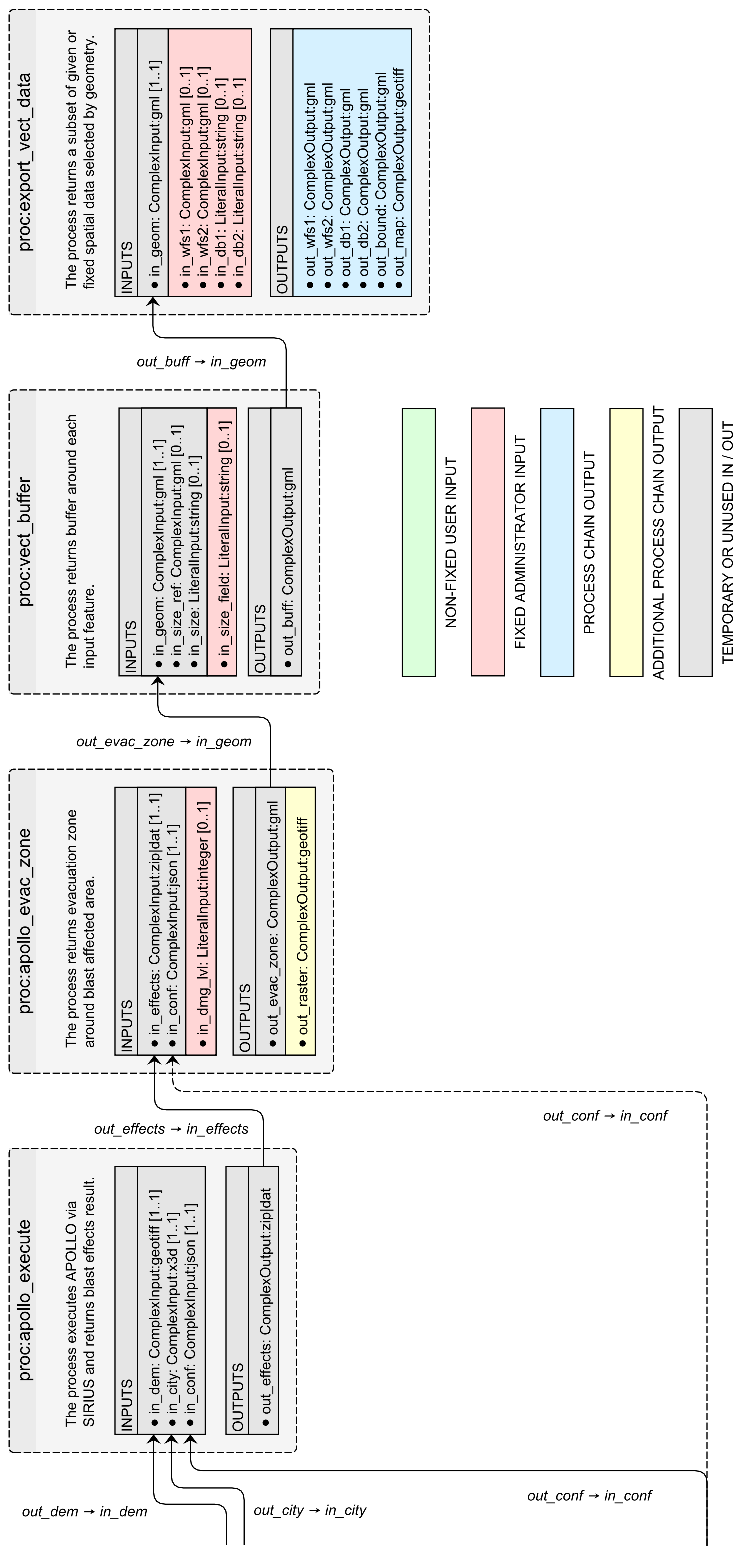

Figure 5.7: Overview of the accurate evacuation zone process chain, part 2

5.5 Key characteristics

The implementation of the processes does not always run smoothly as with the intersection process presented here. During the development of the individual processes, various questions arose or insights were gained that deal with the delimitation, data handling, asynchronous use or chaining of the processes. These questions will be discussed in this section, as they provide some reasons for decisions and ways of implementing the processes. The reasons can be PyWPS bugs, restrictions of the WPS standard or programming style.

5.5.1 Atomicity

If the developed processes are to be reusable for other questions, then special attention must be paid to their delimitation. The more general and abstract the implementation of a process, the greater the probability of reuse for another application. However, this also increases the number of processes and thus the expenditure for development and maintenance. In the context of a municipal SDI the priority is therefore not on the maximum compactness of processes, but on the correct assessment of existing questions (section 3.3), and whether a WPS is suitable for answering them (section 4.1).

The functional delimitation of the various processes was chosen in such a way as to avoid redundancies on the one hand – which increases compactness and modularity, and on the other hand by combining technically similar functions – which reduces compactness and modularity. The result can be described as an individual middle ground, which was achieved through additional and flexibly implemented inputs and outputs. This strategy is especially useful for the four non-case-specific processes, as these are more universally applicable. These processes were based on the following considerations:

- `proc:export_vect_data`: WFS allows the delivery and pre-filtering of geodata from which the selection is to be made. The database support extends the possibilities.
- `proc:export_3d_data`: 3D city model and DEM are often needed together, a single output is also allowed. The combination of the two implemented export processes to one process is possible, but reduces the compactness.
- `proc:vect_buffer`: In addition to a fixed buffer value, this spatial process also supports the name of an input geometry attribute field whose value can be used for a variable buffer size. It only supports the functions that are required in the workflow.

The four case-specific processes, on the other hand, are more difficult to delimit because their intended use serves a specific application, so that the effort and benefit of a high degree of compactness must be weighed up carefully. In the final implementation, the compactness was chosen so that an evacuation zone calculation independent of APOLLO is possible, if a blast effects file is available. Such a separation increases flexibility. However, this decision also has disadvantages, because it requires a redundant call of the configuration process (dashed line in fig. 5.6 and fig. 5.7), respectively for the execute process and the evacuation zone process. It is therefore advisable to consider combining all three APOLLO processes linked in succession to form a single process, but then with renunciation of the mentioned flexibility. However, the rough danger distance process must be outsourced in order to fulfill the requirement of two separate workflows. These processes were based on the following considerations:

- `proc:apollo_rough_dist`: Outsourcing is necessary to meet the requirement of two separate workflows.
- `proc:apollo_conf`: Outsourcing enables the detachment from the subsequent execute process.
- `proc:apollo_execute`: The detachment from the two surrounding processes minimizes the functional limitations in case of network problems, since only this process has to pass through the firewall into the Internet.
- `proc:apollo_evac_zone`: Outsourcing enables the detachment from the previous execute process and thus the calculation of the evacuation zone independent of the APOLLO Blastsimulator. Overall, this process is the most extensive and has a low compactness (fig. 5.4). A separation of certain parts into non-case-specific processes to answer other spatial questions of the city administration of Freiburg has to be carried out if necessary.

5.5.2 Handling of inputs and outputs

In order for a data exchange based on WPS between several processes or a client to function smoothly, special attention must be paid during development to an exact definition of the inputs and outputs. The WPS standard makes certain specifications and defines three data types:

- *LiteralData*: All simple data consisting of a text string or numerical values, i.e. integer, float or string. The parameter `allowed_values` expects a list with which such data can be restricted or predefined.
- *ComplexData*: All non-simple data based on a complex data model, such as raster or vector data. The specification of the appropriate `mime_type` is mandatory. The result of each OGC Web Service (OWS) may also be used as input, which often comes in GML format.
- *BoundingBoxData*: Defines according to the OWS common specification two coordinate pairs in WGS 84 or another reference system by specifying its EPSG code.

The inputs and outputs required for the use case were defined in section 4.3.3. Further rules were established during the implementation to ensure that the data exchange works in practice:

- Some data (location coordinates, APOLLO configuration) are required at different points in the entire process chain. A solution for this can be the use of a workflow engine like Taverna or Camunda BPMN. The looping through of data was avoided.
- Inputs and outputs should be as generic as possible, redundancy-free and serve the purpose of a process, regardless of how the process is used.
- Due to the integration of the PROJ library the reference system for the input geometries is irrelevant, because it is read from the respective data set and transformed if necessary. For the output, on the other hand, the supported reference systems must be clearly described. For all processes of this WPS the processed data are stored in the ETRS89 / UTM zone 32 north (EPSG: 25832) valid in Baden-Württemberg and if required in WGS 84.
- For the exchange of vector data GML, and for raster data GeoTIFF is used.
- Support for optional inputs and outputs increases the versatility of a process. This has been used especially for the vector buffer and the export vector data process.
- Temporarily required files within a process are managed by PyWPS and deleted after the end of the process. It is therefore helpful to use the Python module `tempfile`.
- Writing output data to a database is avoided because there is still no solution for competing processes at database level, which can lead to data loss due to overwriting.

- The use of a geodatabase as input is very performant. The disadvantage, however, is that the structure in the database of the SDI Freiburg has no high semantic interoperability, for example with the names of schemata and database tables. This makes the data exchange between WPS and database very complicated. The use of OWS is more sustainable here.
- Using WFS instead of a geodatabase as input increases flexibility. However, the amount of data to be transferred can increase if no OGC filter encoding is used.

No software without errors, PyWPS is no exception. Also the WPS standard itself has certain shortcomings in the used version 1.0.0. In the case of inputs and outputs, undesirable behaviour occurred in individual cases. There are also disadvantages to some features, such as the use of *RawDataOutput*:

- The use of *RawDataOutput* allows only one output per process, additional outputs are not output. *RawDataOutput* is used by the APOLLO rough danger distance process to output the calculated distance.
- As workaround the vector buffer process additionally uses a *ComplexInput* for the buffer size, because with the PyWPS version used the result of a preceding process can only be read by reference as input.
- PyWPS uses an Universally Unique Identifier (UUID) to distinguish individual process instances. Since WPS standard version 2.0 the JobID was introduced. If, however, processing from components running outside the PyWPS are included, such as the APOLLO Blastsimulator, the problem of competing processes must be managed by these components.
- The use of *BoundingBoxData* is not possible because PyWPS generates a different namespace and XML tag in the output (`ows:BoundingBox`) than is expected in the input (`wps:BoundingBoxData`). This makes chaining impossible.
- Inputs can only be mandatory or optional. There is no possibility to assign two inputs with the condition “either or”.
- Using the data type *float* for a *LiteralOutput* causes a PyWPS error. Switching to the *string* data type fixes the problem.
- The validation mode for *ComplexInput* cannot be used due to incompatibilities between the `mime_type` library and the QGIS WPS client.

5.5.3 Synchronous versus asynchronous

The WPS standard supports two modes in which a process can be executed: synchronous and asynchronous. In synchronous mode, the server accepts the request with the input data and processes it accordingly. During this time, the server waits until the end of the calculations and then returns the resulting process response to the client. In asynchronous mode, the server immediately issues a *ProcessAccepted* response and closes the connection to the client. The process continues to run in the background on the server. The client can check the progress via an offered status URL. After the process is finished and the client requests the status the next time, the final response with the calculation results is output via the status URL. The client itself must be active, because the server only responds to requests and behaves passively. (Čepický and Sousa, 2016)

The asynchronous mode thus enables the execution of long-running processes. This should be used for a duration of 30 seconds or more, because after this time the Apache web server can cause a timeout error depending on the configuration. The APOLLO execute process and the APOLLO evacuation zone process take much longer and are therefore executed asynchronously. All other processes need only a few seconds and are executed synchronously. The APOLLO execute process triggers the actual explosion simulation and can last several hours by use of an accuracy of less than one meter. The use of asynchronous mode must be enabled in the Python code and in the XML request, and the XML parsing must be extended:

- Python process class: Within the *super* function of the process class, the variables `store_supported` and `status_supported` must be set to `True`.
- XML request: In the *ResponseDocument* tag the attribute `storeExecuteResponse` and `status` must be set to `true`. For information the attribute `mode` should be set to `async` in the *Execute* tag.
- XML parsing: To pass on the results to a subsequent process, the status URL must be determined and read every few seconds (`time.sleep(5)`) via a loop. Only if the XML tag *ProcessSucceeded* exists the loop is left.

Starting with version 2.0 of the WPS standard, three additional optional operations are available in asynchronous mode: *GetStatus* to query the status of an asynchronously executed process, *GetResult* to query the result of an asynchronously completed process, and *Dismiss* to terminate an asynchronously started process by the client.

5.5.4 Single use and chained processes

The chaining of processes takes place in the XML request of the *Execute* operation, in which the input of the following process refers to the preceding process and executes this via XML request (listing A.14). An example process chain consisting of intersection and buffer process demonstrates the procedure (fig. 5.8):

> XML request on GitLab:
> https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/chain_sync_vect_intersect_vect_buffer.xml

> XML response result on GitLab:
> https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/chain_sync_vect_intersect_vect_buffer_response.xml

Figure 5.8: Simple intersection and buffer process chain, visualized with QGIS

The use case EOD showed from the beginning that to answer the question by means of WPS a concatenation of several processes would be necessary. Methods have been developed which prepare all processes for use within a process chain. While the basic structure of the chaining of processes as XML request clearly results from the WPS standard, the implementation of the processes must be modified in a few points:

- XML parsing: To pass on the results to a subsequent process, the XML response must be searched for the *ProcessOutputs* tag and parse the individual outputs. The two methods `get_output` and `get_output_data` within the XML parsing library are responsible for this.

- By reference workaround: The result of an preceding process in a chain can only be read by reference as input with the PyWPS version used. Therefore the vector buffer process uses a *ComplexInput* for the buffer size as workaround.
- Asynchronous mode: All final process chains take longer than 30 seconds to process and are therefore executed asynchronously (section 5.5.3).

It becomes apparent that within a process chain only one output of a process can be requested per input of the subsequent process. Therefore, for example, the export 3D related spatial data process must be called twice within the accurate evacuation zone process chain. This makes handling more difficult and the requirements on a client increase. The following is a summary of the challenges found in the course of chaining processes on the basis of WPS:

- Within a process chain, only one output of a process can be requested per input of the following process.
- Intermediate results of processed data within a process chain cannot be assigned to the final total output of the chain. Each process in a chain knows nothing about the chain itself or that it is part of it. Intermediate results are physically on the server, but are not part of *ProcessOutputs*.
- QGIS as WPS client does not yet support WPS process chaining. For this the WPS client plugin would have to be extended with the functionality of a QGIS process provider. Then WPS processes could be chained with the graphical modeler of QGIS.

In summary, it can be said that a close look at the chaining of inputs and outputs reveals the advantages of a workflow engine. Plain XML requests do not provide the flexibility required to reuse processed data at multiple points within the process chain during runtime. This is necessary when using the APOLLO configuration process output, which is required in both the execute process and the evacuation zone process. An optimization by APOLLO would be to extend the header of the blast effects file with the parameters used in the simulation, but in practice it must be assumed that external software components are not easily adaptable.

For the multiple use of non-fixed user inputs before the start of a process chain, a simple HTML form is also technically possible, which distributes the inputs to the respective process inputs via JavaScript. However, this does not solve the problem of redundancy-free reuse of already processed data within the process chain during runtime. Furthermore, a desktop GIS cannot be integrated into an HTML form.

5.6 Limitations for productive operation

The selected use case from the field of EOD is a current research topic and includes external components that are currently under development and may be subject to minor changes. The most important of these components is the unfinished Java Servlet, which acts as a front-end interface between Freiburg's SDI and the APOLLO Blastsimulator, and which will be completed in the second quarter 2019 within the SIRIUS project. This will read the JSON file created by the configuration process and start the APOLLO with the optimal parameters. Only when the Java Servlet is finished this function can be tested practically.

All WPS processes, the applications required for running them and the system-related components, such as the Apache web server, run on a virtual Suse Linux Enterprise Server (SLES) provided by the City of Freiburg. This server is classified as a test system and cannot be reached from the Internet. The provision of the geodata required by APOLLO for external interfaces, such as the Java Servlet, and thus real-time execution of the accurate evacuation zone process chain is therefore not yet possible. For this reason the APOLLO execute process simulates the delivery of the blast effects result file on the own server (listing 5.13).

The APOLLO Blastsimulator is currently being extended by a model for the simulation of splinter throwing, which will further increase the accuracy of the hazard analysis if all site and bomb parameters are known. This model will also be completed in the second quarter 2019 and was not yet available for this master thesis.

During the implementation, great importance was attached to getting as close as possible to real-time execution of the process chain. The missing parts are the responsibility of the SIRIUS project partners. Any adjustments in the Python source code of the affected WPS processes are largely prepared.

CHAPTER 6

Evaluation

6.1 Results of the case study

Taking into account the limitations mentioned, the two independent partial workflows developed in section 4.3.1 for the geodata-related part of an EOD were implemented and successfully tested using a process chain based on the WPS standard. The results processed will be evaluated in this section and compared with the previous procedure. The input data for both process chains are taken from the parameters of the EOD case of 2016 (table 4.1). The initial estimation of the TNT quantity blast power for the rough danger distance process is 400 kg. QGIS was used to locate the coordinates of the site. All user input refers to the non-fixed user inputs (fig. 5.5, fig. 5.6 and fig. 5.7). In addition, all fixed administrator inputs were selected according to table 6.1.

Table 6.1: Input data fixed by the administrator, both process chains

Fixed Administrator Input	**Value or URL**
`apollo_rough_dist:in_solid`	0 (*Float Glass*)
`apollo_evac_zone:in_dmg_lvl`	0 (*Float Glass*, values $1-6$ used for test purposes)
`vect_buffer:in_size_field`	`corr_buff` (attribute field name in evacuation zone GML)
`export_vect_data:in_wfs1`	`http://stadtplan.freiburg.de/wfs7/gdm_poi/poi_public?service=wfs&version=2.0.0&request=getfeature&typename=pois&srsname=epsg:25832`
`export_vect_data:in_wfs2`	Same URL as for `WFS1`, limited by an OGC filter to day-care facilities, police, fire brigade, hospitals, schools and old people's meeting centres.
`export_vect_data:in_db1`	`address` (based on ALKIS, no persons due data privacy)
`export_vect_data:in_db2`	`building` (based on ALKIS, no persons due data privacy)

6.1.1 Process chain output

The quick preselection process chain took about 25 seconds to complete all processes in it. In total, 730 buildings, 499 addresses, 32 general and 6 critical POI are located within the 383 m preselection radius (fig. 6.1). The objects classified as critical include four daycare facilities and two schools.

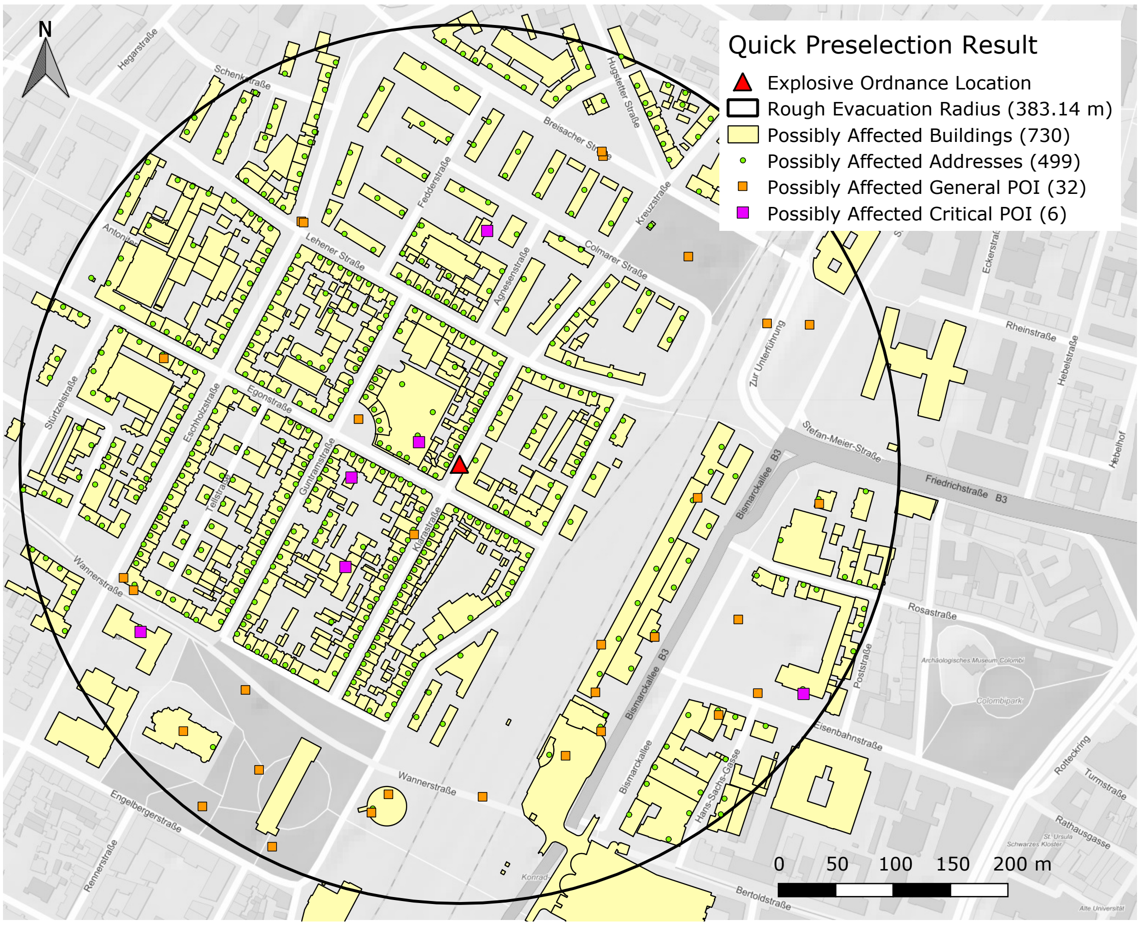

Figure 6.1: Quick preselection result map based on the EOD case 2016

When this EOD case 2016 was processed there was no preselection. The work was performed manually and collected by employees from different departments. Therefore, the integration of this process chain into the entire workflow alone is an added value. However, the currently available POI have a weak point because they do not contain old people's homes or industry. A maintenance of these data and the integration of these in Freiburg's SDI must be managed by the team of the SDI. All data generated by this process chain are available on GitLab (table 6.2).

Table 6.2: Processed data from the quick preselection process chain (* final output)

Output	Value or URL
apollo_rough_dist: out_rough_dist	383.14 m, based on 400 kg TNT
vect_buffer: out_buff	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/quick/out_buff_pre.gml
export_vect_data: out_wfs1*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/quick/out_wfs1_poi_all.gml
export_vect_data: out_wfs2*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/quick/out_wfs2_poi_critic.gml
export_vect_data: out_db1*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/quick/out_db1_address.gml
export_vect_data: out_db2*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/quick/out_db2_building.gml
export_vect_data: out_bound*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/quick/out_bound.gml
export_vect_data: out_map*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/quick/out_map.tif

The accurate evacuation zone process chain took about 120 seconds to complete all processes in it. Excluded from this is the APOLLO execute process, because it was only indirectly linked in the case study. The runtime of APOLLO with an Intel XEON E5 of the Fraunhofer EMI (2.9 GHz, 16 cores) was 40 minutes at a resolution of 1 meter. The simulated time interval is defined as the maximum overpressure until it falls below a critical amplitude and lasts 0.75 seconds for this case. To verify the results, a second simulation with a resolution of 0.5 meters was performed, which lasted 5.5 hours. In total there are 278 buildings, 159 addresses, 2 general and 3 critical POI within the 7.07 ha evacuation zone (fig. 6.2). The critical objects include three daycare facilities.

The evacuation zone of the EOD case 2016 was manually selected based on experience and includes an area of 18.03 ha + 6.50 ha = 24.53 ha. The evacuation zone calculated with APOLLO covers an area of 7.07 ha and is thus much smaller than with the manual method. This reduces the area of buildings and facilities to be evacuated by 71.18 %. For example, it can be seen that the main station would not have had to be evacuated according to the selected parameters. In case of doubt, a look at the evacuation grid (fig. 6.4), which has also been processed, helps whether an estimate value below 0.50 would hit the main station building. The mentioned POI problem remains. All data generated by this process chain are available on GitLab (table 6.3).

Table 6.3: Processed data from the accurate evacuation zone process chain (* final output)

Output	Value or URL
apollo_rough_dist: out_rough_dist	249.15 m, based on 110 kg TNT
vect_buffer: out_buff	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_buff_pre.gml
export_3d_data: out_dem	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_dem.tif
export_3d_data: out_city	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_city.x3d
apollo_conf: out_conf	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_conf.json
apollo_execute: out_effects	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_effects.zip
apollo_evac_zone: out_evac_zone	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_evac_zone.gml
apollo_evac_zone: out_raster	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_raster_f4.tif
vect_buffer: out_buff	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_buff_zone.gml
export_vect_data: out_wfs1*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_wfs1_poi_all.gml
export_vect_data: out_wfs2*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_wfs2_poi_critic.gml
export_vect_data: out_db1*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_db1_address.gml
export_vect_data: out_db2*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_db2_building.gml
export_vect_data: out_bound*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_bound.gml
export_vect_data: out_map*	https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/main/out_map.tif

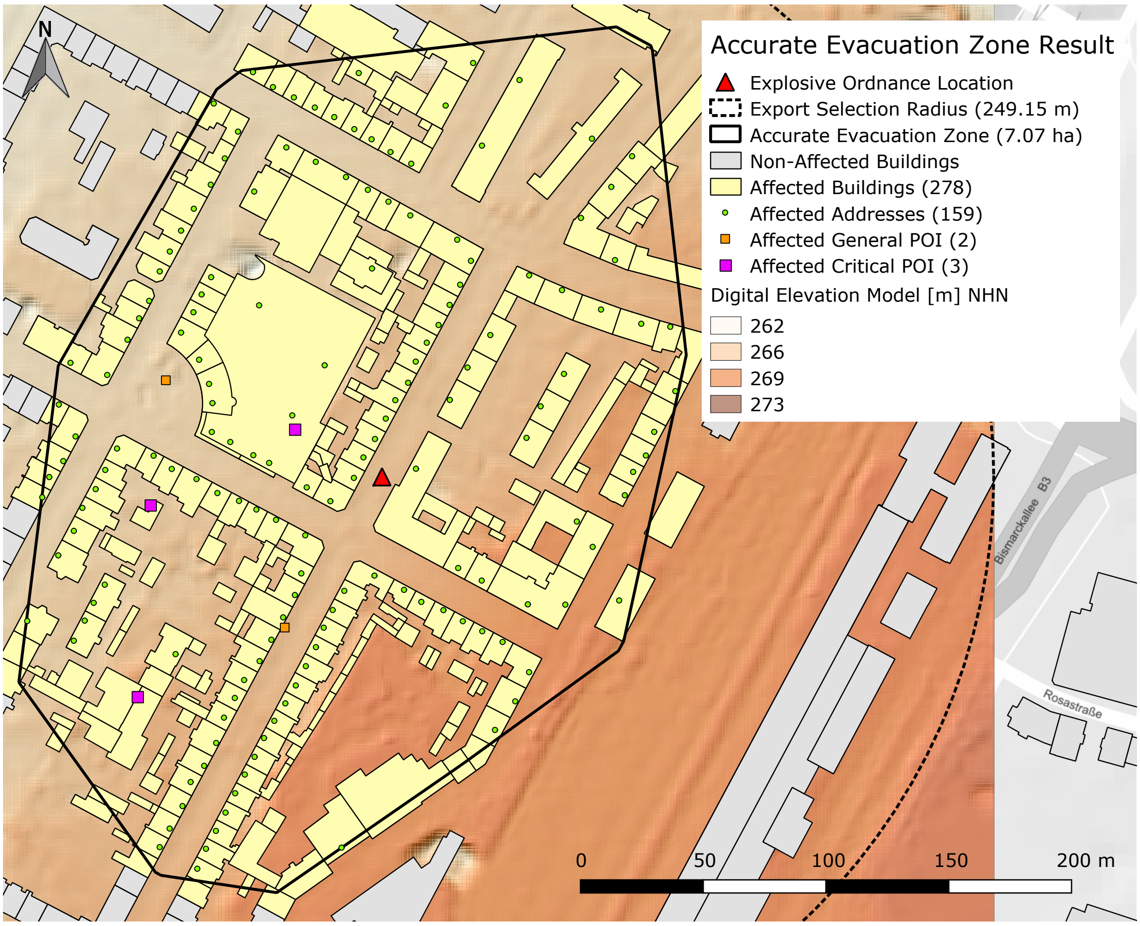

Figure 6.2: Accurate evacuation zone result map with DEM based on the EOD case 2016

6.1.2 Intermediate output

Apart from the main output, further data are processed within the process chain which are necessary for the process but which do not primarily interest the end user. Decisive for the execution of the APOLLO Blastsimulator are the DEM (fig. 6.2) and the 3D city model (fig. 6.3) as data source for the relevant area as well as the simulation parameters in the form of a JSON file (listing 6.1). In addition, the evacuation zone process generates an evacuation grid during the evaluation of the blast effects file, which contains all maximum estimate values in the vertical direction and can be consulted by the expert in case of doubt. Likewise the evacuation zone process can calculate further special danger zones and thus help with the stationing of the emergency forces. For example a narrower zone for police officers with protective suits or the defusing experts within the death zone (fig. 6.4, fig. 6.5, fig. 6.6 and fig. 6.7).

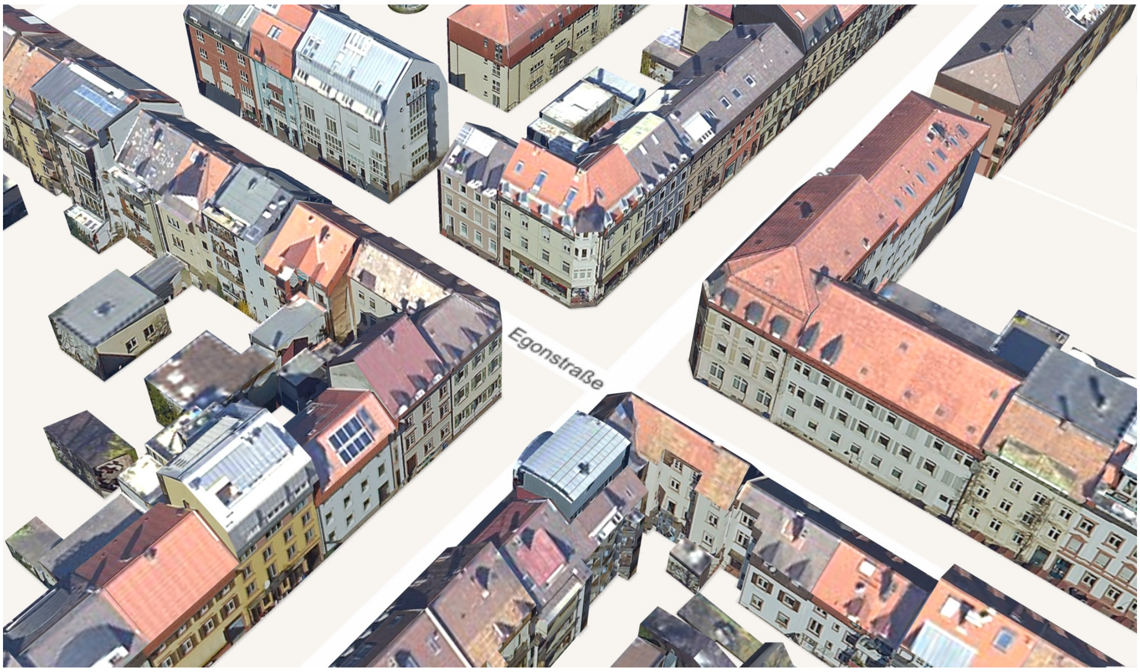

Figure 6.3: Affected district as 3D city model

```json
{
  "crs": 25832,
  "extent": [
    413229.1279899657, 5316613.730901043,
    413727.4356551003, 5317112.038566177
  ],
  "position": [ 413478.281822533, 5316862.88473361 ],
  "height": -2.7,
  "pitch": 0,
  "heading": 0,
  "bomb": { "tnt": 110, "type": "GP250", "detonator": "Front" },
  "site": { "type": "Cavern", "radius": 1.5 },
  "geometry": {
    "crs": 4326, "depth": 2.7,
    "position": [ 7.840131140308953, 47.999206585002355 ]
  },
  "service": {
    "url": "https://www.cadfem.de/apollo/",
    "resultFile": "effects_35cb2598-676c-11e9-8f2e-005056820f34.zip"
  },
  "domain": { "droi": 249.15383256726474, "zroi": 100, "name": "Ultimo" },
  "mode": { "t": 50, "name": "Ultimo", "precision": 1 }
  "hiddenObjects": [ "None" ],
}
```

Listing 6.1: JSON file generated by the APOLLO configuration process

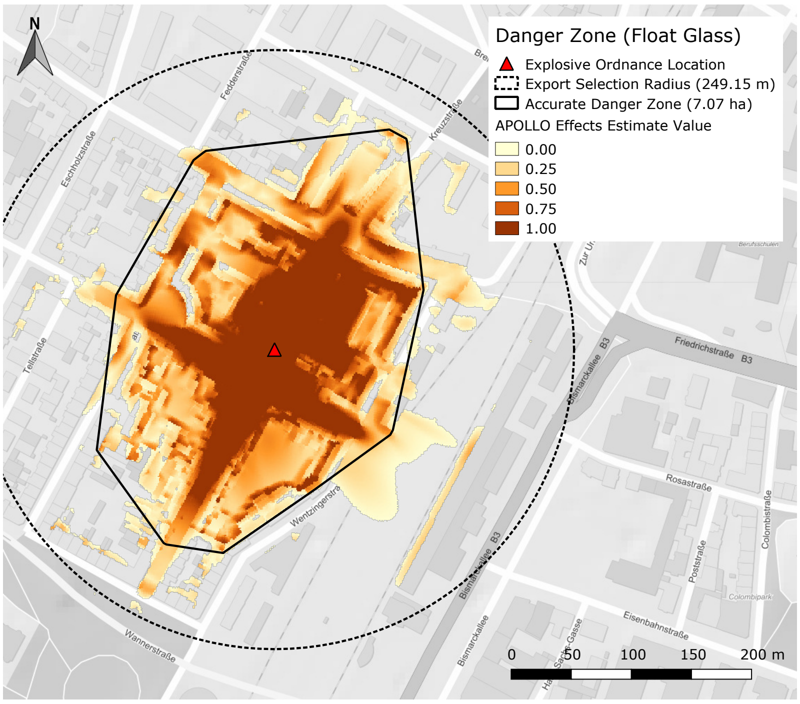

Figure 6.4: All estimate values based on the *Float Glass* characteristic

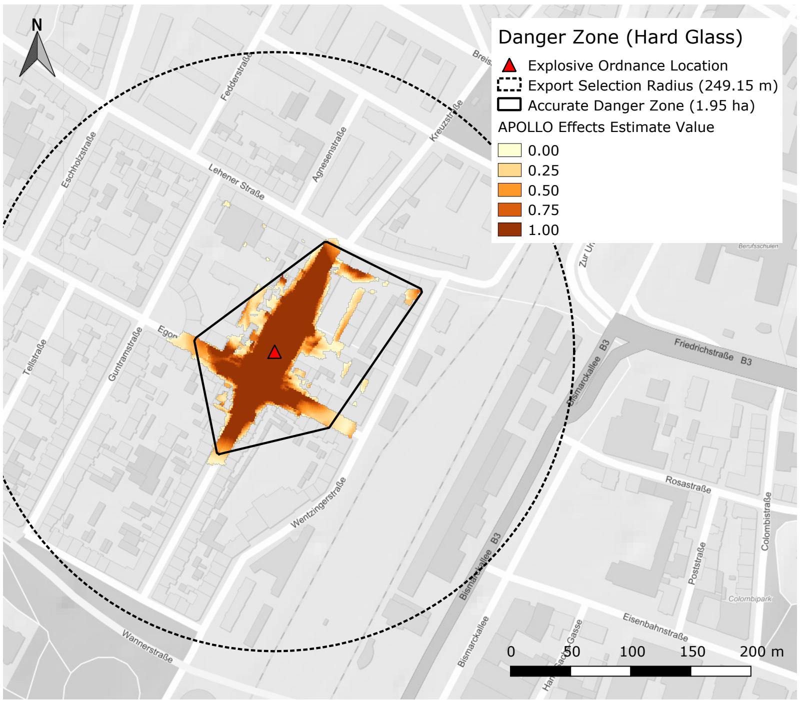

Figure 6.5: All estimate values based on the *Hardened Glass* characteristic

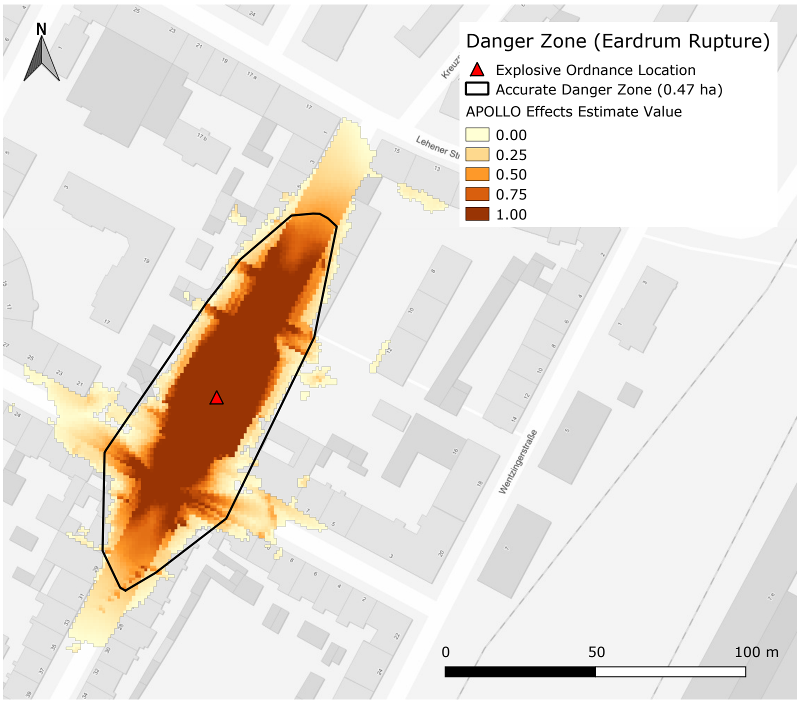

Figure 6.6: All estimate values based on the *Eardrum Rupture* characteristic

Figure 6.7: All estimate values based on the *Lethal Injury* characteristic

6.1.3 Assets and drawbacks

The case study selected for the examination of the applicability of WPS aims to improve the workflow in the evacuation planning in case of an EOD and to minimize the effort for the user. At the same time, the implementation touches different, also non-technical aspects of digitization. The potential improvements identified in section 4.2.2 could be implemented as follows:

1. Integration of the APOLLO Blastsimulator to improve the accuracy of the evacuation radius, the time required for it and the reduction of the dependence on a destruction estimation expert:
   - Taking the limitations (section 5.6) into account, the integration of the Blastsimulator into the selected workflow has increased the accuracy of the evacuation zone and significantly reduced the area to be evacuated.
   - The required time of a few hours is difficult to compare as a manual estimation depends on the availability and experience of a detonation expert. In this respect, the advantage of APOLLO lies in its higher availability and independence from experts.
   - The additional raster danger zones resulting from the simulation are a good help for the differentiated designation of various danger zones. This form of support has not existed at all until now.
2. Use of SDI to facilitate access to the data sets needed and to shorten the time taken to make enquiries to other departments.
   - The use of processes based on WPS now enables a direct connection of the workflow to Freiburg's SDI and thus to the main source of municipal geodata.
   - Through automation in the form of a process chain, many of the required data are available almost immediately and are as up-to-date as in the SDI.
   - At the same time, less specialist staff from different departments is involved in obtaining information and the risk of errors due to outdated data records is reduced.

The improvements mentioned make clear that the optimization of the chosen workflow, including the local SDI, also corresponds to the goals of the digital strategy, so that interdisciplinary processes are harmonized and automated (section 2.1). However, in the context of the case study, individual problems were also identified which could not finally be solved within this master's thesis, but for which initial approaches were considered:

- Data quality and quantity: Processes and digital workflows can only be as good as the data they need. In the case of the official POI it turned out that no old people's homes and no industry are included. The selection of critical infrastructure still has to be completed manually. In the short term, the addition of OSM data[1] (`social_facility` and `social_facility:for` tags) can be recommended, which is more extensive in urban areas (Barron et al., 2014). In the medium term an own WFS with all critical objects and largely based on official data would be conceivable.
- Sensible privacy data: The integration of resident registration data and the city statistics database is not permitted for reasons of privacy. The technical development and digitization precedes the current legal situation, so that it can hardly keep up with the adaptation of the laws (Martini et al., 2016). So it is good to have created another use case that increases the pressure on the legislation.
- Intermediate output: The raster danger zones classified as useful for additional risk assessment cannot be passed on in their present form to the overall output of the process chain. This requires a component that manages the individual inputs and outputs at a higher level than XML, for example a workflow engine.

The current shortcomings of the implementation of the chosen scenario can be summarized by these three points. Also missing model data in development areas are a problem. It is worth mentioning that the apparent hurdle of data protection and privacy can at the same time also be an advantage, because the clever concatenation of processes enables a more precise selection and more targeted delivery of only the actually required protected data. Compared to the previous manual method, the use of a process chain means that fewer people come in touch with sensitive data because the data is output directly to the authorized endpoint. An endpoint does not necessarily have to be a human being at this point, but can also be a technical component. This means that data protection would be completely outside human access, as long as the system is sufficiently secure.

1 https://wiki.openstreetmap.org/wiki/Key:social_facility (visited on 27/04/2019)

6.2 Applicability analysis for WPS

The case study has shown that the elaborated workflow, using WPS processes implemented for it, represents an added value for the actors involved. This section will now abstract the general applicability of such processes for a local SDI, like that of Freiburg. The following criteria were defined in section 3.4, with which the applicability is now to be empirically evaluated.

6.2.1 Reusability

The effort of developing WPS processes is higher compared to a conventional script because the WPS standard and its implementation (e.g. PyWPS) sets certain constraints to the developer. The criterion of reusability examines the additional benefit of such processes, whether they can be used beyond a concrete use case and thus justify the additional effort. It is essential to keep a certain degree of compactness (section 5.5.1) and to define inputs and outputs as generic as possible (section 5.5.2). This can be achieved by keeping in mind the common questions (section 3.3) in a city administration during the designing and programming of the processes. The following is an evaluation of the general reuse potential of all processes, the reuse of single processes as well as the reuse in a process chain. Conventional scripts are usually case-specific and not fully reusable.

Potential for process reuse?

Do at least two of the processes developed for the case study have a higher general potential for reuse?

Reusability is primarily interesting for all non-case-specific processes. But also the case-specific processes should be reusable for slightly modified or similar questions, e.g. from the field of EOD. Before testing the reusability for concrete use cases, a critical overview of the general reuse potential of the implemented processes, including the intersection process, is given (table 6.4). The support for the asynchronous mode, the chaining of processes, as well as the ability for optional outputs applies to all processes and is not mentioned again. The estimation of the potential is mainly based on the following characteristics:

- Generality and versatility of the inputs and outputs of the process (fig. 5.5, fig. 5.6 and fig. 5.7). For example, the GML format is common and allows the use of QGIS.
- Actual reusability within the EOD use case if used more than once.

- Estimated reusability in common questions of a city administration (section 3.3, except again for evacuation radii) for which the process is theoretically possible.
- Availability of alternatives: If suitable alternatives are available and if they are more user-friendly than the implemented process, then the reuse potential of the examined process is limited to the additional use in a process chain.

Table 6.4: Potential of the reusability of the implemented processes (**high**, **moderate**, **low**)

Process	Pro Arguments	Contra Arguments
`vect_intersect`	GML format well-known; after minor adjustments 1x reusable for daily up-to-date intersections for the BZBE;	output only as geometry collection; no handling of attributes; standard operation, suitable alternatives widely available;
`vect_buffer`	GML format well-known; attribute based buffering; all input geometry types supported; 3x used within the EOD use case; 1x reusable for building radio systems;	no handling of attributes, but easy to implement; standard operation, suitable alternatives widely available;
`export_vect_data`	GML format well-known; versatile selection geometry; handling of attributes; WFS as data source and database support; no user-friendly alternatives available; 2x used within the EOD use case; 1x reusable for data delivery;	database use limited to specific topics, but possible to extend; multiple calls necessary if more data sources are required;
`export_3d_data`	GML format well-known; versatile selection geometry; handling of attributes; for city model data no user-friendly alternative available; 1x reusable for data delivery;	city model data output only as X3D; for DEM data any WCS client as alternative available;
`apollo_rough_dist`	simple handling of *LiteralData*; no alternatives available; 2x used within the EOD use case; use independent of APOLLO;	case-specific process, use only for EOD related cases;
`apollo_conf`	GML format well-known; simple handling of *LiteralData*; no alternatives available;	case-specific process, use only for EOD related cases;
`apollo_execute`	no alternatives available;	case-specific process, use only for EOD related cases; APOLLO required for use;
`apollo_evac_zone`	GML format well-known; no alternatives available; use independent of APOLLO;	case-specific process, use only for EOD related cases;

As shown in table 6.4 all non-case-specific processes have a higher reuse potential than case-specific processes. Of the non-case-specific processes, those that answer a complex, frequently asked question or can be considered as part of a process chain have the highest potential. The estimation is strongly dependent on the common questions of a city administration. Any process whose potential is estimated as *high* or *moderate* may be considered as an added value in terms of reusability. Taking into account the low potential of the case-specific processes, the overall estimate of the reuse potential is *moderate*.

Process reusable for a given question?

Is at least one of the processes developed for the case study practically reusable for one of the questions mentioned under section 3.3?

As stated in table 6.4 all non-case-specific processes are reusable for one of the following questions: data delivery, building radio systems and daily up-to-date intersections for the BZBE. These processes work exactly as described in section 5.2 and section 5.3, only the inputs have to be selected depending on the question. The following examples reveal whether at least one of the common questions can be answered in practice with the help of the available processes, or whether further adjustments of the processes are necessary:

- Data delivery: Both export processes are excellent for delivering data sets for a specific area. The use in QGIS simplifies the creation of the required selection geometry. Frequently used geodata, such as 3D city models, addresses, buildings, parcels or local plans are already implemented. The addition of further geodata is easily possible. The practical reusability of `export_vect_data` is demonstrated on the basis of a fictitious request of an engineering office for legally binding local plans in the Freiburg district *Altstadt* (fig. 6.8).
- Building radio systems: The buffer process can be very well reused for this question and can be easily integrated into the existing workflow thanks to QGIS. The required attribute based buffering is supported by the process and can be preselected as fixed input for simplification. The practical reusability of `vect_buffer` is demonstrated by three buildings with different levels of building radio systems (fig. 6.9).
- Intersections for the BZBE: If support for object attributes is added, the intersection process can be reused for this question. However, the problem of missing semantic interoperability of the geodata in the database has a negative effect, which increases the effort. A stronger use of WFS is recommended.

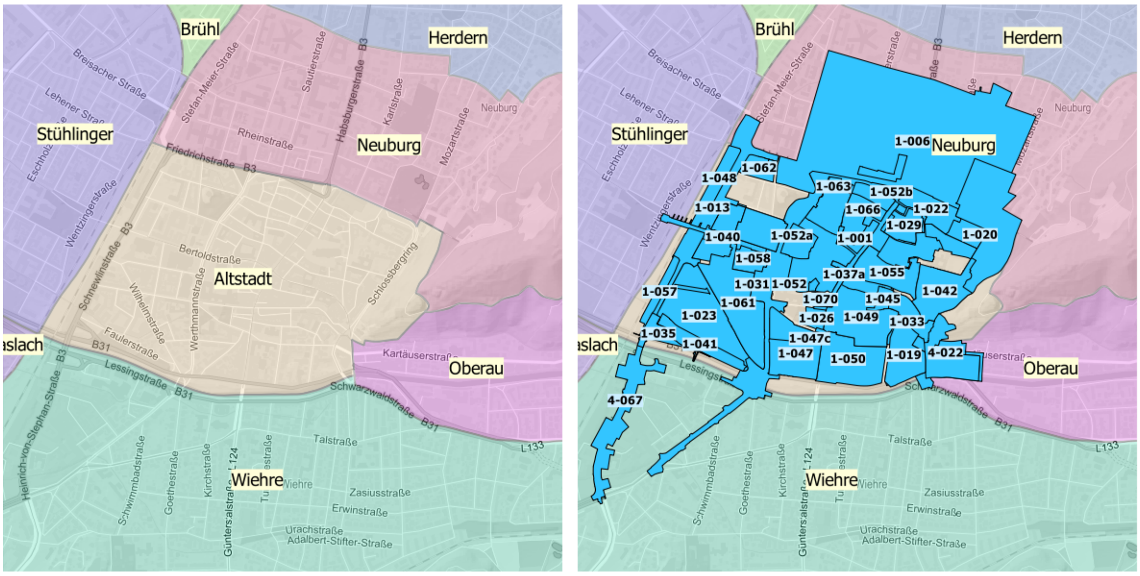

Figure 6.8: Reusability of `export_vect_data` using the example of local plans

> Selection district on GitLab: https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/misc/district.gml
> Selected local plans on GitLab: https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/misc/local_plans.gml

Figure 6.9: Reusability of `vect_buffer` using the example of building radio systems

> Building radio systems on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/misc/build_radio.gml`
> Buffered buildings on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/data/misc/build_radio_buff.gml`

Of the three questions considered, two could be answered with the available processes from the EOD workflow. An adaptation of the processes was not necessary for these two scenarios. This corresponds to 33 % of the common questions described in section 3.3. A minor modification of the intersection process would also be able to answer the third selected question from the BZBE. Due to the large number of procedures in a city administration, it can be assumed that further workflows can be implemented with just a few additional processes. Because of the standardized WPS interface, the combination possibilities increase with the number of available WPS processes. This also increases the probability of reusability, which in turn can lead to a higher number of WPS using workflows. According to this logic, the number of processes will increase faster at the beginning, and slower once a pool of processes exists. The required reuse of at least one process was exceeded, therefore the overall estimate of the reusability for one of the given questions is *fulfilled*.

Create more than one process chain?

Is it possible to use the available processes to create another process chain of at least two processes to answer a question?

As described in section 2.1 and as the implemented EOD workflow shows, complex questions are often to be found in a city administration. Often such a complexity is not realizable with a single process, therefore the use of chained processes in a local SDI is to be classified as important. The workflow for evacuation planning in the case of an EOD shows another concatenation of three processes: the quick preselection chain. Thus this criterion was sufficiently fulfilled with a chain of at least two processes. But in general it has to be said that complex processes often contain a component that is only needed for this specific procedure. In the EOD use case the APOLLO Blastsimulator is such a component, managed by the case-specific processes, without which both process chains would not be realizable. For the data maintenance of the daycare facilities mentioned in section 2.1 and section 3.3 this component could be the BKG geocoding service, managed by a process for automated integration into the workflow. In addition, the number of implemented processes is currently small. Therefore a further process chain for a question outside the EOD is not feasible with the available processes, if thereby a workflow with added value in the sense of the digitization is to develop. Of course, chains can be formed from the available processes, for example a combination of intersection process and export vector data process. But such an application is rare, often individual, and easier to realize with conventional processing. In the sense of reusability, the criterion of another chain of at least two processes was *fulfilled*.

6.2.2 Compatibility

The criterion of compatibility examines the interoperability and adaptability of WPS processes in interaction with the heterogeneous IT structure of a city administration in general and its SDI with the corresponding components in particular. Likewise this criterion is an important prerequisite for a good reusability, and thus the sustainability of the solutions based on WPS, as well as for a good technical usability. For conventional scripts any compatibility must be implemented more or less costly by yourself.

Compatibility with added value?

Can the existing components of the SDI be used by a WPS with added value?

Due to the standardization it can be assumed that a WPS is basically compatible with an OGC compliant SDI. But in practice a high compatibility alone does not automatically lead to a high added value, because it depends on how exactly the OGC standards are used and how advanced the SDI is in its structure. Based on the experiences from the case study, an evaluation of the compatibility and the added value of the SDI components used will be given and summarized in table 6.5.

- PostgreSQL with PostGIS: The connection is made via free program libraries like *Psycopg* or OGR, therefore compatibility is basically given. However, a lack of semantic interoperability of the geodata in the database increases the effort for generic access to this data. This contradicts the striving for generic inputs and outputs of WPS processes. An example from the case study: For each geodata table exported from the database a separate SQL query is necessary. If the database structure changes, these SQL queries must be maintained in a time-consuming manner. A direct connection to the database is only recommended if the required data is not available as an OGC service, or if the amount of data is so high that the better performance of the database is an argument, e.g. 3D data. The storage of

WPS processed results as source for OGC services is possible, but the problem of competing processes has to be considered. The processes `export_vect_data` and `export_3d_data` read the database. The compatibility and the added value are estimated as *moderate*.

- QGIS with WPS client: The use of a WPS is possible with restrictions in QGIS. As demonstrated (section 5.1) single WPS processes can be executed if QGIS supports the inputs. The use as workflow engine for chaining processes is not yet implemented in the WPS client. Except for `apollo_execute` and `apollo_evac_zone`, because of the blast effects file, all processes including `apollo_conf` are supported by QGIS. Due to the lack of process chain support, compatibility and added value are estimated as *moderate*.

- Web Map Service (WMS): The compatibility between OGC services is high as expected. However, the use of WMS for a processing service offers little added value. A grid without further semantics provides only little information relevant for urban processing. Of all implemented processes, the `export_vect_data` process uses a WMS as an additional output to display a topographic map. This corresponds to 0.5 % of all WMS provided by the SDI. The compatibility is estimated as *high*, the added value as *low*.

- Web Feature Service (WFS): Geodata requested via a WFS can be returned as vector data and are therefore well suited for answering urban questions by means of WPS, because these often happen on the actual geometries and rarely on the raster level. Of all implemented processes the `export_3d_data` process uses WFS as input, whose geodata is then selected and exported. Technically, any WPS process that supports *ComplexInput* can use WFS as generic input, which makes it very versatile. In this master thesis the buildings, area boundaries, POI and local plans were used as WFS with the implemented processes. This corresponds to 20 % of all WFS provided by the SDI. The compatibility and the added value are estimated as *high*.

- Web Coverage Service (WCS): Unlike WMS, a WCS provides multidimensional coverage data based on the original data set, with full semantics for machine processing. Such services are not very common in a city administration and can be used well without a WPS, for example directly in QGIS. Of all implemented processes, the `export_3d_data` process uses a WCS to provide an extract from the DEM. This corresponds to 100 % of all WCS provided by the SDI. The compatibility is estimated as *high*, the added value as *moderate*.

- Catalogue Service for the Web (CSW): Finding the processes provided by WPS can be simplified by registering the service in a CSW. The created WPS has been successfully registered via GeoNetwork and can be found via a CSW client. Should a WPS process be implemented for citizens or external service providers, which is not impossible in terms of digitization in the urban context, it could also be found from outside the local SDI. Moreover, this would counteract the mentioned lack of sources for finding WPS (section 1.1). However, the high specialization of processes in the urban context, as the EOD workflow shows, makes them poorly usable for people outside the city administration, for example because of missing access rights. The compatibility is estimated as *high*, the added value as *moderate*.

Table 6.5: Overview of the components used with their compatibility and added value

Component	Compatibility	Added Value
PostgreSQL with PostGIS	moderate	moderate
QGIS with WPS client	moderate	moderate
WMS 1 of 200 used	high	low
WFS 4 of 20 used	high	high
WCS 1 of 1 used	high	moderate
CSW	high	moderate

It can be said that a WPS benefits the most from an SDI based on OGC standards, and an intensive use of WFS brings the highest efficiency. In addition, a WFS comes closest to the goal of the intensive use of generic inputs for a robust, widely usable data exchange with WPS processes. Freiburg makes too little use of this and here lies the great potential of its SDI. The compatibility with QGIS and the geodatabase is important in everyday life, and can be increased by further development on the client side and by improving the semantic interoperability on the database side. The question whether the components of the SDI can be used by a WPS with added value is answered with *moderate*.

Adaptability?

Is the adaptability of a WPS sufficient to support the heterogeneous IT structure of a city, such as by integrating previously unintegratable technical procedures?

Adaptability is achieved on the one hand by the standardized WPS interface (section 2.4), and on the other hand by the possibilities of the WPS implementation itself, like PyWPS in this case. The case study has shown how APOLLO was connected and made usable as a component that is actually not compatible with the SDI Freiburg. The following properties of WPS were used to achieve this adaptation:

- Definition of any inputs and outputs as *ComplexData*. Only with this capability arbitrary, even proprietary data formats can be used by a process. This is proven in the EOD workflow by the blast effects file, which is used in the APOLLO execute process as output, and in the APOLLO evacuation zone process as input.
- Using asynchronous mode for the long-running APOLLO execution process and the APOLLO evacuation zone process.
- Use of the capability to chain processes to map the complexity of the EOD workflow.
- The use of a Python-based WPS implementation offers the possibility to use all packages available in Python for adaptation to other components. In the implemented processes for example the packages JSON, OSGeo, *NumPy* or *Psycopg* are used.

The APOLLO Blastsimulator from the case study demonstrates, representative of many other more or less specific components, that WPS and its processes are adaptable enough to be integrated into a workflow. The flexibility of WPS ensures that previously incompatible, non-integratable components can be connected to an open SDI and thus benefit from the advantages of this SDI. The connection is realizable independently of the will of a software manufacturer, provided that an open, documented, readable exchange format is supported, as the proprietary APOLLO blast effects file shows. However, it must be considered that a proprietary component usually requires a WPS process that is not reusable for other cases, which illustrates the four case-specific processes, and which increases the effort. The adaptability of a WPS and its processes to support a heterogeneous IT structure is estimated as *high*, similar to the adaptability of a conventional script.

Use of external WPS?

Can the functionality of a WPS capable SDI be extended by externally provided processes?

The standardization of the WPS interface basically enables the accessibility of all WPS within a network, and thus the integration of externally provided processes into own procedures. The use of an externally provided WPS is especially suitable for standard processing like intersection, buffer, contains or distance. These can reduce the effort for the implementation of complex WPS based workflows, because in the ideal case only case-specific processes have to be developed. The development of an own buffer and intersection process, as in this thesis, could be omitted, and both processes would still be available for a process chain within the local SDI. Now two WPS providers will be tested for the accessibility of their processes. In addition, the example of a buffer process is used to theoretically check whether it can replace the buffer process developed for the EOD workflow. A practical check is not possible due to the limitations mentioned (section 5.6), because a process chain consisting of internal and external processes requires the accessibility of the test system from outside.

Terrestris: This company offers a WPS which contains 58 spatial processes. A comparison shows that their buffer process does not support attribute based buffering and is therefore not suitable for the EOD workflow. In QGIS the process could be called correctly, but it did not accept the selected geometry layer as input and acknowledged it with an *InvalidParameterValue* error message. For verification the same request was sent again using the *RESTClient* and ended with a similar error message.

> Request on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/terrestris_buffer.xml`
> Response on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/terrestris_buffer_response.xml`
> Service provider tested on 12/05/2019:
> `https://ows.terrestris.de/deegree-wps/services?request=DescribeProcess&service=WPS&version=1.0.0&identifier=Buffer`

52° North: Another provider of a WPS with 221 geoprocessings. Their WPS offers a simple and a complex buffer process. The simple one provides only two parameters for the input and the buffer size. The complex one supports the attribute based buffering required for the EOD workflow. The check in QGIS was acknowledged with a Java exception message and names the input parsing as cause. Using a completely different geometry layer or checking it again with the *RESTClient* produces an identical result.

> Request on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/52north_buffer.xml`
> Response on GitLab: `https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/52north_buffer_response.xml`
> Service provider tested on 12/05/2019:
> `http://geoprocessing.demo.52north.org:8080/wps/WebProcessingService?request=DescribeProcess&service=WPS&version=1.0.0&identifier=v.buffer`

Basically the integration of external processes is a good possibility to extend the functionality of a WPS capable SDI, if these processes meet the requirements. But in practice there is a shortage of availability of such processes (Lopez-Pellicer et al., 2012), and differences in the realization of inputs cause compatibility problems between WPS and WPS client. Therefore, this criterion is estimated as *low*, but with potential.

Any other side effects?

Does a WPS have any other side effects in terms of compatibility?

During the implementation of the EOD workflow, opportunities and dependencies were identified that could have positive or negative effects on the implementation of further WPS processes. Some of them lead to recommended actions. This criterion is not suitable for a rating. The following points should be mentioned here:

- Motivation to provide more WFS: As mentioned, WFS is little used so far. This is due to the use of QGIS as main tool for geoprocessing, which is directly connected to the database. The integration of WPS into the SDI of Freiburg provides a good reason to increase the offer of WFS in order to be able to use the advantages mentioned and shown in the case study, such as a higher semantic interoperability of the geodata. In addition to its better suitability for WPS, a WFS can also be used for reading without a detailed rights structure. This is more difficult to handle on the database and therefore an additional added value for the administration of the SDI.
- Legal compatibility: The case study has shown that the legal framework lags behind technological development in certain areas. The topic of sensitive data and data privacy is affected by this. The residents' registration data and the statistics database were not allowed to be integrated into the EOD workflow. Therefore, ways must be found to describe how sensitive data should be handled in terms of automation and digitization, and how they should be protected, but also how they can be used.

6.2.3 Usability

The two previous criteria have mainly examined the conceptual and technical properties of WPS based processes and whether they meet the requirements of local SDIs. However, the acceptance by administrators and users is also decisive for a successful use, which is evaluated with the criterion of usability. Here the characteristics of a city administration mentioned in section 3.4.3 must be taken into account.

Administration usability?

The evaluation of the technical usability includes the handling and the possibilities of the concrete WPS implementation. The necessary system and programming skills are a prerequisite. However, it can be assumed that a city administration the size of Freiburg usually does not employ any studied computer scientists in the SDI team.

Effort of integrating a WPS?

- The preparation of an environment for providing a WPS should not be underestimated, because components like web server and operating system have to be adapted to the used WPS implementation. Likewise the knowledge about the basic structure of a process must be acquired. This effort has to be done only once and is therefore negligible, because once the principle is understood, simple processes can be implemented quickly.
- In a city administration, standard processes that are relatively easy to implement, such as buffer or intersection, are often only an addition to a more complex workflow. If specific components from outside the GIScience are used, the knowledge about each additional component has to be acquired anew. The EOD workflow with integration of the APOLLO Blastsimulator is a good example. This is the price for the high flexibility, therefore the effort of integrating a new WPS process is estimated as *high*.

Effort of adjusting and maintaining a WPS?

- Basically, the effort for adapting and maintaining existing processes is somewhat less than with a script, because the basic structure in the source code always remains the same. For example, processes can be extended with optional inputs or outputs without having to change existing requests and workflows. Likewise, reusability ensures that changes to the actual processing only have to be adapted in one place.
- At the same time, any change must be very carefully considered, as the effects are greater the more often a process is reused. This increases the effort for conceptual work and testing of all affected use cases. Due to the precision in the definition of inputs and outputs required by the standard, the susceptibility to errors decreases when used properly. Thanks to the same basic structure, troubleshooting is always more efficient than in scripts with a non-uniform structure. The effort of adjusting and maintaining existing WPS processes is estimated as *moderate*.

Additional effort for the chaining of processes?

- Due to the required precision in the definition of inputs and outputs, the effort for mapping a workflow into a process chain increases. All subprocesses must be chainable, support asynchronous mode due to the often longer processing time, and have to be parsed (section 5.5.4). The orchestration of a process chain with pure XML is limited and not very flexible. With conventional scripts, a process chain based on standardized interfaces is hardly feasible.
- A workflow engine like Taverna or Camunda BPMN enables the request of multiple outputs of a process and their assignment to corresponding inputs of subsequent processes. With SOAP and WSDL WPS supports the necessary standards for integration into industry standard service chaining tools. Intermediate results of processed data within a process chain can be assigned to the total output. The XML parsing no longer has to take place within the processes. Many of the disadvantages identified in the EOD workflow can be eliminated with a workflow engine, and justify a practical follow-up check of such a one. The additional effort for chaining WPS processes is estimated as *high*.

Possibilities of simplification for the users?

- WPS offers the possibility to preassign inputs and to request outputs only on demand. This avoids incorrect inputs during operation, which is especially important for safety-relevant workflows. The EOD use case demonstrates this by preassigning *Float Glass* as fixed input for the rough danger distance and evacuation zone process (table 6.1), and thus prevents the accidental calculation of an evacuation zone unsuitable for citizens, for example for *Safety Glass* or *Lethal Injury*. A drawback is the lack of support for logical constraints, such as assigning “either or” to two inputs.
- If a process is supported by QGIS, there are many possibilities for simplification. By creating a QGIS template, required geodata and tables can be preconfigured for the respective workflow. Setting conditions and adapting forms guides the user and minimizes incorrect inputs. Outside of QGIS, the effort for such simplifications increases, especially if geodata is required for an input. For example, via an HTML form that distributes the data via JavaScript to the respective process inputs, with leaflet for simple geometries. The operation of WPS process chains could thus be simplified, but without the features of a complete workflow engine. The possibilities of simplification for the users are estimated as *moderate*.

The high effort in the orchestration of WPS processes to a workflow must be compared to the permanent time saving in comparison to the previous manual solution. Many workflows require repetitive activities on the part of the administrators, if these are not automated, and are in the sum more time-consuming than the implementation of a process chain. Regular data delivery is an example of this. Compared to conventional scripts, the time savings are lower. Here the reuse and chaining of components as well as the somewhat smaller effort for adjusting and maintaining is an advantage of WPS, especially with regard to the increase of digital workflows in public administrations. Based on the experiences made with the EOD use case and the mentioned arguments, the technical usability of WPS and its processes is estimated as *moderate*.

Application usability?

The evaluation of usability, which concerns users in dealing with WPS processes developed by the system operator, covers the specific use case, with which effort and in which quality a question can be answered. Basic computer skills are a prerequisite, which is well reflected in reality by the decreasing average age of public sector employees. Nevertheless, the requirements for users are higher compared to a conventional script due to the extended possibilities.

Availability of the WPS?

- The principle of an SOA ensures that a WPS can be used by anyone connected to the same network. This makes workflows less dependent on individual persons or expensive computer-bound software licenses, and guarantees personnel reliability as well as more flexible working conditions. The EOD use case shows how only one instance of the APOLLO Blastsimulator is sufficient and can be used by multiple persons. This way of deployment also enables integration into clients, giving each user the ability to use available processes where they need them. Finding a WPS is simplified by an entry in a metadata catalog and can be done by any CSW enabled client.

- The availability of the data required for each processing is usually managed by the local SDI and its connected components. Thus a high topicality is reached and guarantees to the user the use of the most current data sets, as the EOD use case demonstrates at the example of DEM, POI, addresses or 3D city model. The availability of a WPS is estimated as *high*, especially compared to a manual workflow.

Need for clients and special software?

- The functionality to trigger individual processes or entire workflows is provided by WPS clients and is the part with which the user comes into direct contact. The task is to distribute all required data to the respective inputs of the processes, to start the processing and to receive the final result. Ideally, the client can also prepare the data, which means a seamless transfer of the data for the user.
- QGIS offers itself as a client (fig. 6.10) for spatial processes, which can work with data formats from the GIScience and process geodata extensively. All important OGC standards and geodatabases are supported, which simplifies the use of the connected SDI. Six of eight implemented processes can be operated directly. Due to the high functionality of this client, the flexibility, but also the susceptibility to errors in the data preparation is higher. QGIS is not suitable for the execution of process chains, only the input data can be prepared.
- Due to the heterogeneous IT structure of a city administration, a client for non-spatial data must also be available. With an Internet browser the execution of process chains can be simplified, for example via the mentioned HTML form. To avoid many of the chaining problems identified in the EOD workflow and to simplify the assignment of inputs and outputs, a workflow engine is required (section 5.5.4). WPS clients are often freely available, but they require a certain amount of training for the user. The available clients and the need for special software is estimated as *moderate*.

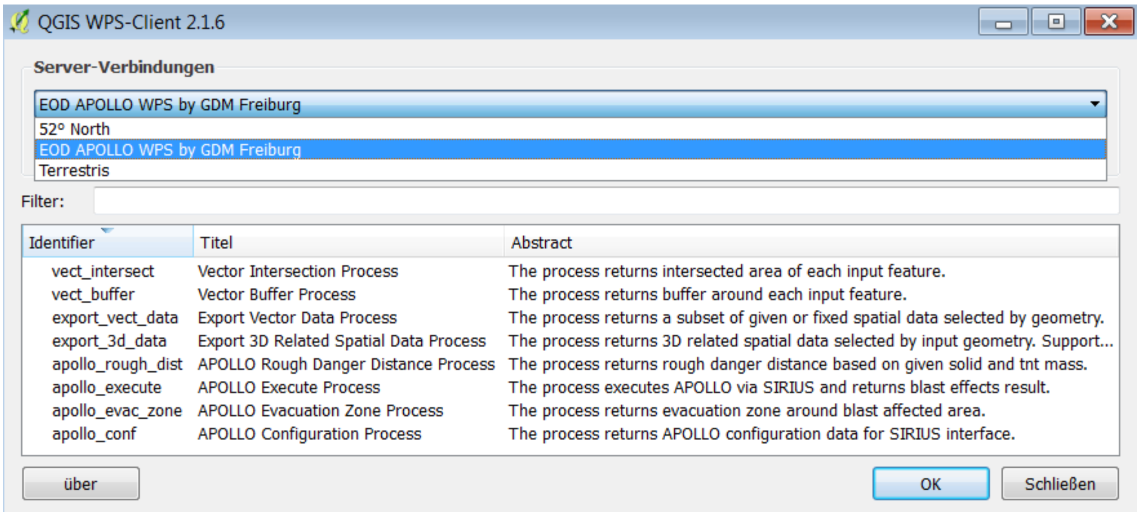

Figure 6.10: QGIS as WPS client for single use processes

Effort of answering a question?

- The effort to answer a question includes the preparatory steps of the user as well as the time until the answer is available. The time required depends on the complexity of the processing and is significantly shorter than with a manual workflow due to automation and reduction of the number of actors involved. The APOLLO Blastsimulator is an extreme example, because the calculation tasks in a city administration are usually less complex, and external service providers are rarely integrated into automated processes. The more people have been involved in a process so far, the greater the potential time saving in the future. Waiting times due to understaffing or busy offices are eliminated because manual intervention is no longer necessary during processing.
- The elimination of actors can increase the effort required for preparatory steps because, depending on the application, many more decisions required for a process chain have to be made by a single person. However, this depends strongly on the concrete implementation of a workflow. Preparatory steps are usually clearly described and can be easily carried out by the user, because the results must be accepted by the process inputs and then processed without errors, and must also be operable by persons outside the GIScience. For example, the processing of required geodata in QGIS or the triggering of a process chain via a simple website. Nevertheless, compared to a conventional script, the effort required to answer a question is higher, because until now it has been limited to filling out an HTML form or pressing a button. But in comparison to a manual workflow, the total effort required by the user is significantly lower and therefore estimated as *low*.

The usability in the application of a WPS depends on the individual process, and becomes more difficult when using a process chain. The case study has shown that QGIS can serve a wide range of processes. For all other processes and process chains solution approaches were shown. Decisive for the user is the comparison with his previous approach, which is different for each question. If the advantages predominate, the complexity remains hidden, and the handling of the new workflow is trained, a broad acceptance is realistic. Based on the experiences made with the application of the EOD use case and the mentioned arguments, the usability for the operators of WPS and its processes is estimated as *moderate*.

CHAPTER 7

Conclusion and outlook

The increasing need to automate municipal operations using the local SDI has raised the question of a uniform approach. The literature research resulted in possible solution approaches, but which did not specifically address the applications and requirements in a municipal SDI. Therefore the investigation of the applicability of WPS processes in a local SDI based on open standards was the focus of this master thesis.

In order to test the hypothesis common questions were pointed out, which have to be answered by a city administration, ideally using their SDI. A concrete use case was selected, which refers to the geodata-relevant part in the planning of an evacuation in the case of an EOD. The implementation of the use case as a process chain based on WPS integrates an external component from outside the GIScience in order to investigate the applicability of WPS in a realistic way and with inclusion of the heterogeneous IT structure of a city administration. The following evaluation is based on the findings of the final EOD workflow, abstracted on a local SDI using the example of the city of Freiburg and three selected criteria: reusability, compatibility, usability. The summary of the criteria evaluated in section 6.2 shows the areas in which the advantages and disadvantages of WPS lie when used in a local SDI, and how the equivalents of conventional scripts used in Freiburg approximately perform (table 7.1).

The direct comparison with conventional scripts is often not possible, because the respective approaches are too different. Nevertheless, it must be mentioned that especially a good usability of WPS process chains is more difficult to achieve than that of a script. Also, the adaptability of a WPS is not better, but roughly equal to the flexibility of a script. Furthermore, conventional scripts are implemented faster because the conceptual phase is less complex and there are fewer dependencies.

Table 7.1: Summary of the criteria and their grades for WPS and conventional scripts

Criterion	**WPS**	**Scripts**
	Evaluation of Reusability	
Potential for process reuse?	moderate	failed
Process reusable for a given question?	fulfilled	failed
Create more than one process chain?	fulfilled	failed
	Evaluation of Compatibility	
Compatibility with added value?	moderate	low
Adaptability?	high	high
Use of external WPS?	low	failed
Any other side effects?	not suitable for rating	not suitable for rating
	Evaluation of Usability	
Administration usability?	moderate	moderate
Application usability?	moderate	high

The generally known advantages of WPS are primarily the interoperability with each other and with other OGC services. This is accompanied by the reusability and eventual reduction of development costs, as well as hiding the complexity of components. A large part of these advantages can also be transferred to the use in a local SDI, however with few limitations and some peculiarities:

- Reusability: Due to many proprietary components in a city administration, the probability of reuse of individual processes can decrease slightly, as the four case-specific processes from the EOD workflow show. Also the conceptual delimitation between the processes takes place based on the correct assessment of existing questions. The goal is not the maximum compactness or the supply for general, unknown use cases, but the purposeful reuse for own, known and common questions.
- Compatibility: The added value of WPS using an open SDI depends on the use of the available standards and the level of semantic interoperability of the data, as well as on the quality of the data itself. Therefore, an intensive use of WFS is advantageous for the common questions in a city administration. It has also turned out that the integration of highly sensitive data can lead to a legal impasse due to data protection, but WPS is also an opportunity to minimize access to such data. Especially in a local SDI such data are of importance for many processes.

- Usability: The wide range of different specialists within a city administration increases the demands on application usability. This is not automatically given and must be established, either using compatible clients or a workflow engine. The latter seems to be of great advantage for the integration of a process chain, since process chains are particularly suitable for mapping the often complex processes in a city administration.

The entire study has shown that the application of WPS processes in a local SDI has led to a significant added value. This becomes visible above all by the criterion of reusability, because the processes implemented for the EOD workflow can be reused for two other use cases without additional development effort. Such a reuse is not covered by a script at this level. This confirms the hypothesis. However, the amount of added value depends on how intensively WPS will be used in the future. With increasing number of WPS based workflows also the number of processes increases, and thus the probability for a reusability. Decisive for this is also the continuing development of the local SDI towards a larger range of available WFS. Due to the interoperability, besides a mutual added value also other components can benefit from it, which harmonizes well with the development of the SDI. From this point of view, the increased initial effort for the implementation of WPS processes in comparison to conventional scripts can be justified additionally. The initial effort is individually different, but must not be concealed.

The EOD workflow itself has also led to a significant qualitative improvement compared to the previous approach to evacuation planning. The identified disadvantages are above all the incomplete data to the critical infrastructure, the prohibition of the integration of sensitive data, as well as the restricted handling of the process chain regarding the access to intermediate results. Due to the high specialization it will be difficult to obtain knowledge of such a WPS outside the borders of the local territorial authority. But if the offer of WPS from higher regional authorities or scientific institutes should increase and become generally usable, a local SDI could benefit very well from standard processes. Conversely, the demand for freely accessible and general standard processes would increase if WPS were used more widely, which in turn would promote the research field of standardization of geoprocessing itself. Whether the arguments and added values found in this master thesis are sufficient to permanently use WPS in local SDIs will be shown by further practical applications. The applicability of WPS in a local SDI has been demonstrated using the example of the city of Freiburg and verified by means of a real use case.

Regardless of the knowledge gained, there is still a need for further research on the applicability of WPS for local SDIs. The GitLab repository will still be available. Relevant questions are an in-depth investigation of suitable WPS clients and workflow engines in the context of a city administration, as well as the practical application of these to a wide range of common questions in the communal area. Concerning the EOD workflow, the data on critical infrastructure should be completed and the results of the SIRIUS project integrated. With the integration of the Java Servlet, the workflow can be executed in real time, and with the extension of the APOLLO Blastsimulator for the simulation of splinter throwing, a further increase in the accuracy of evacuation radii can be expected.

Bibliography

Altmaier, A. and M. Müller (2002): 'Geodateninfrastrukturen in der Praxis'. *Zeitschrift für Angewandte Geographie*, vol. (3): pp. 103–106 (cit. on pp. 9, 10).

Barron, C., P. Neis, and A. Zipf (2014): 'A Comprehensive Framework for Intrinsic OpenStreetMap Quality Analysis'. *Transactions in GIS*, vol. 18(6): pp. 877–895 (cit. on p. 77).

Bartelme, N. (2005): *Geoinformatik - Modelle, Strukturen, Funktionen*. Springer Berlin Heidelberg (cit. on p. 10).

Bettenworth, M. (2013): *Verfahren zur Analyse von Detonationseinwirkungen in urbanen Gebieten (DETORBA)*. Bundesministerium für Bildung und Forschung. url: https://www.sifo.de/de/detorba-verfahren-zur-analyse-von-detonationseinwirkungen-in-urbanen-gebieten-2094.html (visited on 03/02/2019) (cit. on p. 29).

Brennecke, C. (2015): 'Geoverarbeitung im Wandel: Vom Desktop in das World Wide Web'. Master Thesis. Paris Lodron-Universität Salzburg (cit. on p. 2).

Čepický, J. (2019): *PyWPS API Doc*. PyWPS API Doc. url: https://pywps.readthedocs.io (visited on 02/18/2019) (cit. on p. 44).

Čepický, J. and L. M. d. Sousa (2016): 'New implementation of OGC Web Processing Service in Python programming language. PyWPS-4 and issues we are facing with processing of large raster data using OGC WPS'. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLI-B7: pp. 927–930 (cit. on p. 64).

Gebhard, A. (2018): *Simulationsbasierte Gefährdungsanalyse im urbanen Raum für Einsätze des Kampfmittelräumdienstes (SIRIUS)*. Bundesministerium für Bildung und Forschung. url: https://www.sifo.de/de/sirius-simulationsbasierte-gefaehrdungsanalyse-im-urbanen-raum-fuer-einsaetze-des-2318.html (visited on 03/02/2019) (cit. on p. 29).

Giuliani, G., S. Nativi, A. Lehmann, and N. Ray (2012): 'WPS mediation: An approach to process geospatial data on different computing backends'. *Computers & Geosciences*, vol. 47: pp. 20–33 (cit. on p. 12).

Hahmann, S. and D. Burghardt (2012): 'Forschungsergebnisse zur Frage: Haben 80% aller Informationen einen Raumbezug?' *gis.SCIENCE*, vol. (3): pp. 101–108 (cit. on p. 8).

Hermsdorf, J. (2016): *Sichere Deflagration von Blindgängern durch Lasertechnologie (DEFLAG)*. Bundesministerium für Bildung und Forschung. URL: https://www.sifo.de/de/deflag-sichere-deflagration-von-blindgaengern-durch-lasertechnologie-2282.html (visited on 03/02/2019) (cit. on p. 29).

Hofer, B. (2015): 'Uses of online geoprocessing technology in analyses and case studies: a systematic analysis of literature'. *International Journal of Digital Earth*, vol. 8(11): pp. 901–917 (cit. on pp. 1, 13).

Hogrebe, D. (2008): 'Mehrwertschöpfung durch Integration von OpenGIS in kommunale Prozesse'. Master Thesis. Paris Lodron-Universität Salzburg (cit. on p. 2).

Kinney, G. F. and K. J. Graham (1985): *Explosive Shocks in Air*. Springer Berlin Heidelberg (cit. on pp. 29, 50).

Kliment, T. (2015): 'Making more OGC services available on the web discoverable for the SDI community'. 15th International Multidisciplinary Scientific GeoConference (cit. on p. 9).

Klomfass, A., N. Kirchner, O. Herzog, S. Knell, V. Holzwarth, U. Ziegenhagel, and M. Sauer (2009): 'C++ Code Design for Multi-Purpose Explicit Finite Volume Methods: Requirements and Solutions'. *Proceedings of the 8th workshop on Parallel/High-Performance Object-Oriented Scientific Computing - POOSC '09*. Genova, Italy: ACM Press: pp. 1–5 (cit. on p. 29).

Klomfass, A., A. Stolz, and S. Hiermaier (2016): 'Improved Explosion Consequence Analysis with combined CFD and Damage Models'. *Chemical Engineering Transactions*, vol. 48: pp. 109–114 (cit. on p. 29).

Königer, S., S. Volz, D.-G. Hielscher, S. Erat, B. Schindewolf, I. Wanders, G. Bär, and P. Geier-Baumann (2017): *Kommunale Pflichtaufgaben beim Aufbau der europäischen Geodateninfrastruktur INSPIRE*. Geoportal Baden-Württemberg. URL: https://www.geoportal-bw.de/documents/20147/0/INSPIRE-Kommunale-Betroffenheit-BW_V2.0_final_20170504.pdf/3f1d072b-5430-3b7e-f68a-1ca0e44a0e00 (visited on 01/17/2019) (cit. on p. 14).

Lopez-Pellicer, F. J., W. Rentería-Agualimpia, R. Béjar, P. R. Muro-Medrano, and F. J. Zarazaga-Soria (2012): 'Availability of the OGC geoprocessing standard: March 2011 reality check'. *Computers & Geosciences*, vol. 47: pp. 13–19 (cit. on pp. 2, 88).

MARTINI, M., S. FRITZSCHE, and M. KOLAIN (2016): *Digitalisierung als Herausforderung und Chance für Staat und Verwaltung*. German Research Institute for Public Administration Speyer (cit. on pp. 7, 8, 77).

SAMADZADEGAN, F., M. SABER, H. ZAHMATKESH, and H. JOZE GHAZI KHANLOU (2013): 'An architecture for automated fire detection early warning system based on geoprocessing service composition'. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-1/W3: pp. 351–355 (cit. on p. 1).

SCHUT, P. (2007): *OpenGIS Web Processing Service*. Open Geospatial Consortium. URL: http://portal.opengeospatial.org/files/?artifact_id=24151 (visited on 12/11/2018) (cit. on p. 11).

SEIFERT, M. (2005): 'Das AFIS-ALKIS-ATKIS-Anwendungsschema als Komponente einer Geodateninfrastruktur'. *zfv - Zeitschrift für Geodäsie, Geoinformation und Landmanagement*, vol. (2): pp. 77–81 (cit. on p. 11).

STASCH, C., B. PROSS, B. GRÄLER, C. MALEWSKI, C. FÖRSTER, and S. JIRKA (2018): 'Coupling sensor observation services and web processing services for online geoprocessing in water dam monitoring'. *International Journal of Digital Earth*, vol. 11(1): pp. 64–78 (cit. on p. 1).

STOLLBERG, B. and A. ZIPF (2007): 'OGC Web Processing Service Interface for Web Service Orchestration Aggregating Geo-processing Services in a Bomb Threat Scenario'. *Web and Wireless Geographical Information Systems*. Ed. by WARE, J. M. and G. E. TAYLOR. Springer Berlin Heidelberg: pp. 239–251 (cit. on pp. 2, 12).

TAN, X., L. DI, M. DENG, F. HUANG, X. YE, Z. SHA, Z. SUN, W. GONG, Y. SHAO, and C. HUANG (2016): 'Agent-as-a-service-based geospatial service aggregation in the cloud: A case study of flood response'. *Environmental Modelling & Software*, vol. 84: pp. 210–225 (cit. on p. 1).

TROMETER, S. (2015): *Abschlussbericht: Verfahren zur Analyse von Detonationseinwirkungen in urbanen Gebieten (DETORBA)*. CADFEM GmbH (cit. on pp. 29, 30).

WALENCIAK, G., B. STOLLBERG, S. NEUBAUER, and A. ZIPF (2009): 'Extending Spatial Data Infrastructures 3D by Geoprocessing Functionality - 3D Simulations in Disaster Management and environmental Research'. *International Conference on Advanced Geographic Information Systems & Web Services*. GEOWS 2009. Cancun, Mexico: IEEE: pp. 40–44 (cit. on p. 1).

YOON, G., K. KIM, and K. LEE (2017): 'Linkage of OGC WPS 2.0 to the e-Government Standard Framework in Korea: An Implementation Case for Geo-Spatial Image Processing'. *ISPRS International Journal of Geo-Information*, vol. 6(1): p. 25 (cit. on p. 1).

A Appendix

A.1 Python source code

A.1.1 PyWPS WSGI instance script

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/pywps.wsgi

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" PyWPS WSGI instance - A wrapper around the PyWPS server with
    a list of processes and configuration files passed as arguments.
"""

# libs
import sys
from pywps.app import Service

# processes need to be installed in PYTHON_PATH
sys.path.append('/srv/www/wps/')

from processes.proc_apollo_conf import ApolloConf
from processes.proc_apollo_evac_zone import ApolloEvacZone
from processes.proc_apollo_execute import ApolloExecute
from processes.proc_apollo_rough_dist import ApolloRoughDist
from processes.proc_export_3d_data import Export3dData
from processes.proc_export_vect_data import ExportVectData
from processes.proc_vect_buffer import VectBuffer
from processes.proc_vect_intersect import VectIntersect

processes = [
    ApolloConf(),
    ApolloEvacZone(),
    ApolloExecute(),
    ApolloRoughDist(),
    Export3dData(),
    ExportVectData(),
    VectBuffer(),
    VectIntersect()
]

# for the process list on the home page
process_descriptor = {}
for process in processes:
    abstract = process.abstract
    identifier = process.identifier
    process_descriptor[identifier] = abstract

# Service accepts list of process instances and list of configuration files
application = Service(processes, ['/srv/www/wps/pywps.cfg'])
```

Listing A.1: PyWPS WSGI instance script

A.1.2 Vector intersection process

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/proc_vect_intersect.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" The process returns intersected area of each input feature.
"""

# libs
import logging
import requests
import tempfile
from pywps import Process, ComplexInput, ComplexOutput, Format
from pywps.app.Common import Metadata
from pywps.validator.mode import MODE
from pywps.validator import complexvalidator
from osgeo import ogr
from osgeo import osr
from lxml import etree
from lib import varlib

# authorship information
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")


# process returns intersected area of each input feature
class VectIntersect(Process):
    def __init__(self):
        in_geom_a = ComplexInput(
            'in_geom_a',
            'Input Geometry A [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml),
                               Format(mime_type='application/gml+xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            mode=MODE.NONE
        )

        in_geom_b = ComplexInput(
            'in_geom_b',
            'Input Geometry B [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml),
                               Format(mime_type='application/gml+xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            mode=MODE.NONE
        )

        out_intersect = ComplexOutput(
            'out_intersect',
            'Intersected Geometry',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
```

```python
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      encoding='UTF-8', validate=None)]
        )

        inputs = [in_geom_a, in_geom_b]
        outputs = [out_intersect]

        super(VectIntersect, self).__init__(
            self._handler,
            identifier='vect_intersect',
            version='1.0',
            title='Vector Intersection Process',
            abstract='The process returns intersected area of each input feature.',
            metadata=[Metadata('The process returns intersected area of each input feature.',
                               'http://geodev:8080/geonetwork/srv/ger/catalog.search?service=CSW&version=2.0.2'
                               '&request=GetRecordById&id=c850b578-8561-42fb-88d1-1ac9e3314cf4#/metadata/'
                               'c850b578-8561-42fb-88d1-1ac9e3314cf4')],
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    # handler method obtains request object and response object
    # @staticmethod  # only for static methods, no 'self' applicable
    def _handler(self, request, response):
        # check if data is given by reference
        if request.inputs['in_geom_a'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_geom_a'][0].method == 'GET':
                # obtain input with identifier as file name
                in_geom_a = request.inputs['in_geom_a'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_geom_a'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_geom_a'][0].data

                LOGGER.debug('XML Response:' + in_response)

                # get content of LiteralData, Reference or ComplexData
                ref_url = varlib.get_output(etree.fromstring(in_response))

                # get GML file as reference
                r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
                data = r.content

                # create file, w: write in text mode
                filename = tempfile.mkstemp(prefix='geom_a_', suffix='.gml')[1]
                with open(filename, 'w') as fp:
                    fp.write(data)
                    fp.close()

                in_geom_a = filename
        else:
            # obtain input with identifier as file name
            in_geom_a = request.inputs['in_geom_a'][0].file

        # check if data is given by reference
        if request.inputs['in_geom_b'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_geom_b'][0].method == 'GET':
                # obtain input with identifier as file name
                in_geom_b = request.inputs['in_geom_b'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_geom_b'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_geom_b'][0].data

```

```
            LOGGER.debug('XML Response:' + in_response)

            # get content of LiteralData, Reference or ComplexData
            ref_url = varlib.get_output(etree.fromstring(in_response))

            # get GML file as reference
            r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
            data = r.content

            # create file, w: write in text mode
            filename = tempfile.mkstemp(prefix='geom_b_', suffix='.gml')[1]
            with open(filename, 'w') as fp:
                fp.write(data)
                fp.close()

            in_geom_b = filename
    else:
        # obtain input with identifier as file name
        in_geom_b = request.inputs['in_geom_b'][0].file

    # open file and layer of input a
    in_src_a = ogr.Open(in_geom_a)
    in_lyr_a = in_src_a.GetLayer()
    lyr_name_a = in_lyr_a.GetName()

    # open file and layer of input b
    in_src_b = ogr.Open(in_geom_b)
    in_lyr_b = in_src_b.GetLayer()
    lyr_name_b = in_lyr_b.GetName()

    # get and set output spatial reference
    epsg = int(in_lyr_a.GetSpatialRef().GetAttrValue('AUTHORITY', 1))
    sref = osr.SpatialReference()
    sref.ImportFromEPSG(epsg)

    # create output file
    driver = ogr.GetDriverByName('GML')
    out_src = driver.CreateDataSource(lyr_name_a)
    out_lyr = out_src.CreateLayer(lyr_name_a+'_'+lyr_name_b, sref, ogr.wkbGeometryCollection)

    # create geometry collection of input a
    collect_a = ogr.Geometry(ogr.wkbGeometryCollection)
    for feat in in_lyr_a:
        collect_a.AddGeometry(feat.GetGeometryRef())

    # create geometry collection of input b
    collect_b = ogr.Geometry(ogr.wkbGeometryCollection)
    for feat in in_lyr_b:
        collect_b.AddGeometry(feat.GetGeometryRef())

    # calculate intersection
    intersect_geom = collect_a.Intersection(collect_b)

    # create output feature to the file
    out_feat = ogr.Feature(feature_def=out_lyr.GetLayerDefn())
    out_feat.SetGeometry(intersect_geom)
    out_lyr.CreateFeature(out_feat)

    # free and reassign
    out_feat = None
    out_src = None

    # set output format and file name
    response.outputs['out_intersect'].output_format = Format(mime_type='text/xml', extension='.gml',
                                                         schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                                         encoding='UTF-8', validate=None)
    response.outputs['out_intersect'].file = lyr_name_a

```

```
        return response
```

Listing A.2: Vector intersection process

A.1.3 Vector buffer process

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/proc_vect_buffer.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" The process returns buffer around each input feature.
"""

# libs
import logging
import requests
import tempfile
from pywps import Process, LiteralInput, ComplexInput, ComplexOutput, Format
from pywps.app.Common import Metadata
from pywps.validator.mode import MODE
from pywps.validator import complexvalidator
from osgeo import ogr
from osgeo import osr
from lxml import etree
from lib import varlib

# authorship information
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")


# process process returns buffer around each input feature
class VectBuffer(Process):
    def __init__(self):
        in_geom = ComplexInput(
            'in_geom',
            'Input Geometry [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml),
                               Format(mime_type='application/gml+xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            # validation mode unable to use due incompatibilities between mimetype library and QGIS wps client
            mode=MODE.NONE
        )

        in_size_ref = ComplexInput(
            'in_size_ref',
            'Buffer Size Reference',
            abstract='Buffer size calculated by previous process only chainable as reference.',
            supported_formats=[Format(mime_type='text/plain')],
            min_occurs=0
        )

```

```
        in_size = LiteralInput(
            'in_size',
            'Buffer Size [m]',
            data_type='string',  # use of string instead float as workaround
            min_occurs=0
        )

        in_size_field = LiteralInput(
            'in_size_field',
            'Buffer Size Field Name',
            abstract='Name of input geometry attribute field which value will be used for buffer size.',
            data_type='string',
            min_occurs=0
        )

        out_buff = ComplexOutput(
            'out_buff',
            'Buffer Geometry',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      encoding='UTF-8', validate=None)]
        )

        inputs = [in_geom, in_size_ref, in_size, in_size_field]
        outputs = [out_buff]

        super(VectBuffer, self).__init__(
            self._handler,
            identifier='vect_buffer',
            version='1.0',
            title='Vector Buffer Process',
            abstract='The process returns buffer around each input feature.',
            metadata=[Metadata('The process returns buffer around each input feature.',
                               'http://geodev:8080/geonetwork/srv/ger/catalog.search?service=CSW&version=2.0.2'
                               '&request=GetRecordById&id=c850b578-8561-42fb-88d1-1ac9e3314cf4#/metadata/'
                               'c850b578-8561-42fb-88d1-1ac9e3314cf4')],
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    # handler method obtains request object and response object
    # @staticmethod  # only for static methods, no 'self' applicable
    def _handler(self, request, response):
        # check if data is given by reference
        if request.inputs['in_geom'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_geom'][0].method == 'GET':
                # obtain input with identifier as file name
                in_geom = request.inputs['in_geom'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_geom'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_geom'][0].data

                LOGGER.debug('XML Response:' + in_response)

                # get content of LiteralData, Reference or ComplexData
                ref_url = varlib.get_output(etree.fromstring(in_response))

                # get GML file as reference
                r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
                data = r.content

                # create file, w: write in text mode
                filename = tempfile.mkstemp(prefix='geom_', suffix='.gml')[1]
                with open(filename, 'w') as fp:
```

```
                fp.write(data)
                fp.close()

            in_geom = filename
        else:
            # obtain input with identifier as file name
            in_geom = request.inputs['in_geom'][0].file

        # default parameter values
        size, size_field = 0, ''

        # check and obtain input with identifier as data directly
        if 'in_size' in request.inputs:
            size = request.inputs['in_size'][0].data
        if 'in_size_field' in request.inputs:
            size_field = request.inputs['in_size_field'][0].data
        if 'in_size_ref' in request.inputs:
            size_ref = request.inputs['in_size_ref'][0].data

            # buffer size priority by reference
            if float(size_ref):
                size = float(size_ref)

        # open file and layer
        in_src = ogr.Open(in_geom)
        in_lyr = in_src.GetLayer()

        # get layer name
        lyr_name = in_lyr.GetName()

        # get all field names of input layer
        field_names = [field.name for field in in_lyr.schema]

        # get and set output spatial reference
        epsg = int(in_lyr.GetSpatialRef().GetAttrValue('AUTHORITY', 1))
        sref = osr.SpatialReference()
        sref.ImportFromEPSG(epsg)

        # create output file
        driver = ogr.GetDriverByName('GML')
        out_src = driver.CreateDataSource(lyr_name)
        out_lyr = out_src.CreateLayer(lyr_name+'_buff', sref, ogr.wkbPolygon)

        # get feature count
        count = in_lyr.GetFeatureCount()
        index = 0

        # make buffer for each feature
        while index < count:
            # get the geometry
            in_feat = in_lyr.GetNextFeature()
            in_geom = in_feat.GetGeometryRef()

            # check if size attribute exists
            if size_field in field_names:
                size_val = in_feat.GetField(size_field)
                if isinstance(size_val, int) or isinstance(size_val, float):
                    size = size_val
                else:
                    size = 0

            LOGGER.debug('Buffer Size:' + str(size))

            # make the buffer
            buff_geom = in_geom.Buffer(float(size))

            # create output feature to the file
            out_feat = ogr.Feature(feature_def=out_lyr.GetLayerDefn())
```

```
            out_feat.SetGeometry(buff_geom)
            out_lyr.CreateFeature(out_feat)

            # free and reassign
            out_feat = None

            index += 1

        # free and reassign
        out_src = None

        # set output format and file name
        response.outputs['out_buff'].output_format = Format(mime_type='text/xml', extension='.gml',
                                                          schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                                          encoding='UTF-8', validate=None)
        response.outputs['out_buff'].file = lyr_name

        return response
```

Listing A.3: Vector buffer process

A.1.4 Export vector data process

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/proc_export_vect_data.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" The process returns a subset of given or fixed spatial data selected by geometry.
"""

# libs
import logging
import tempfile
import requests
import owslib.util
from pywps import Process, LiteralInput, ComplexInput, ComplexOutput, Format
from pywps.app.Common import Metadata
from pywps.validator.mode import MODE
from pywps.validator import complexvalidator
from owslib.wms import WebMapService
from osgeo import ogr
from osgeo import osr
from lxml import etree
from lib import varlib
from lib import geolib

# authorship information
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")


# process returns a subset of given or fixed spatial data selected by geometry
class ExportVectData(Process):
    # static class variables
    epsg = 25832  # local spatial reference code
```

```python

    def __init__(self):
        in_geom = ComplexInput(
            'in_geom',
            'Selection Geometry [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            mode=MODE.NONE
        )

        in_wfs1 = ComplexInput(
            'in_wfs1',
            'WFS Request 1 [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            mode=MODE.NONE,
            min_occurs=0
        )

        in_wfs2 = ComplexInput(
            'in_wfs2',
            'WFS Request 2 [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            mode=MODE.NONE,
            min_occurs=0
        )

        in_db1 = LiteralInput(
            'in_db1',
            'Database Spatial Data Name 1',
            abstract='Supported spatial data is defined by the following names: '
                     'address, building, parcel, local_plan, poi',
            data_type='string',
            allowed_values=('address', 'building', 'parcel', 'local_plan', 'poi'),
            min_occurs=0
        )

        in_db2 = LiteralInput(
            'in_db2',
            'Database Spatial Data Name 2',
            abstract='Supported spatial data is defined by the following names: '
                     'address, building, parcel, local_plan, poi',
            data_type='string',
            allowed_values=('address', 'building', 'parcel', 'local_plan', 'poi'),
            min_occurs=0
        )

        out_wfs1 = ComplexOutput(
            'out_wfs1',
            'WFS Request 1 Subset',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      encoding='UTF-8', validate=None)]
        )

        out_wfs2 = ComplexOutput(
            'out_wfs2',
            'WFS Request 2 Subset',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      encoding='UTF-8', validate=None)]
        )

        out_db1 = ComplexOutput(
```

```
            'out_db1',
            'Database Spatial Data 1 Subset',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      encoding='UTF-8', validate=None)]
        )

        out_db2 = ComplexOutput(
            'out_db2',
            'Database Spatial Data 2 Subset',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      encoding='UTF-8', validate=None)]
        )

        out_bound = ComplexOutput(
            'out_bound',
            'Selection Boundary',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      encoding='UTF-8', validate=None)]
        )

        out_map = ComplexOutput(
            'out_map',
            'Output Data Overview Map',
            supported_formats=[Format(mime_type='image/geotiff', extension='.tif')]
        )

        inputs = [in_geom, in_wfs1, in_wfs2, in_db1, in_db2]

        outputs = [out_wfs1, out_wfs2, out_db1, out_db2, out_bound, out_map]

        super(ExportVectData, self).__init__(
            self._handler,
            identifier='export_vect_data',
            version='1.0',
            title='Export Vector Data Process',
            abstract='The process returns a subset of given or fixed spatial data selected by geometry.',
            metadata=[Metadata('The process returns a subset of given or fixed spatial data selected by geometry.',
                               'http://geodev:8080/geonetwork/srv/ger/catalog.search?service=CSW&version=2.0.2'
                               '&request=GetRecordById&id=c850b578-8561-42fb-88d1-1ac9e3314cf4#/metadata/'
                               'c850b578-8561-42fb-88d1-1ac9e3314cf4')],
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    # handler method obtains request object and response object
    # @staticmethod  # only for static methods, no 'self' applicable
    def _handler(self, request, response):
        # check if data is given by reference
        if request.inputs['in_geom'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_geom'][0].method == 'GET':
                # obtain input with identifier as file name
                in_geom = request.inputs['in_geom'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_geom'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_geom'][0].data

                LOGGER.debug('XML Response:' + in_response)

                # get content of LiteralData, Reference or ComplexData
                ref_url = varlib.get_output(etree.fromstring(in_response))

```

```
                # get GML file as reference
                r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
                data = r.content

                # create file, w: write in text mode
                filename = tempfile.mkstemp(prefix='input_', suffix='.gml')[1]
                with open(filename, 'w') as fp:
                    fp.write(data)
                    fp.close()

                in_geom = filename
        else:
            # obtain input with identifier as file name
            in_geom = request.inputs['in_geom'][0].file

        # open file and layer
        in_src = ogr.Open(in_geom)
        in_lyr = in_src.GetLayer()

        # get spatial reference
        epsg0 = int(in_lyr.GetSpatialRef().GetAttrValue('AUTHORITY', 1))

        # get geometry
        feat = in_lyr.GetNextFeature()
        geom = feat.GetGeometryRef()

        # only one single polygon input feature
        if in_lyr.GetFeatureCount() == 1 and geom.GetGeometryName() == 'POLYGON':
            # harmonization of spatial reference
            if epsg0 != self.epsg:
                # transform selection geometry to spatial reference of 3D city model
                sref0 = osr.SpatialReference()
                sref0.ImportFromEPSG(epsg0)
                sref = osr.SpatialReference()
                sref.ImportFromEPSG(self.epsg)
                transform = osr.CoordinateTransformation(sref0, sref)
                geom.Transform(transform)

            LOGGER.debug('Input Geometry of Type ' + str(geom.GetGeometryName()) +
                         ' in ' + str(self.epsg) + ':' + geom.ExportToWkt())

            # WFS PART ##################################################

            if 'out_wfs1' in request.outputs.keys():
                # check and obtain input with identifier as data directly
                if 'in_wfs1' in request.inputs:
                    wfs1 = request.inputs['in_wfs1'][0].data

                    # create file, w: write in text mode
                    in_path = tempfile.mkstemp(prefix='wfs1_data_', suffix='.gml')[1]
                    with open(in_path, 'w') as fp:
                        fp.write(wfs1)
                        fp.close()

                    LOGGER.debug('WFS1 Data String:' + str(wfs1[0:1000]))

                    # open file and layer
                    wfs1_src = ogr.Open(in_path)
                    wfs1_lyr = wfs1_src.GetLayer()

                    # get spatial reference
                    wfs_epsg = int(wfs1_lyr.GetSpatialRef().GetAttrValue('AUTHORITY', 1))

                    LOGGER.debug('WFS1 Feature Count in ' + str(wfs_epsg) + ':' + str(wfs1_lyr.GetFeatureCount()))

                    # check spatial reference
                    if wfs_epsg == self.epsg:
                        wfs1_lyr.SetSpatialFilter(geom)
```

```
                else:
                    LOGGER.debug('Incompatible Spatial Reference of WFS1 and Selection Geometry.')

                LOGGER.debug('WFS1 Feature Count After Filter:' + str(wfs1_lyr.GetFeatureCount()))

                # set output format definition
                out_path = tempfile.mkstemp(prefix='wfs_' + wfs1_lyr.GetName() + '_data_', suffix='.gml')[1]
                out_src = ogr.GetDriverByName("GML").CreateDataSource(out_path)
                out_src.CopyLayer(wfs1_lyr, wfs1_lyr.GetName())

                # set output format and file name
                response.outputs['out_wfs1'].output_format = Format(mime_type='text/xml', extension='.gml',
                                                                    schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                                                    encoding='UTF-8', validate=None)
                response.outputs['out_wfs1'].file = out_path
        else:
            # remove output from response
            del response.outputs['out_wfs1']

        if 'out_wfs2' in request.outputs.keys():
            # check and obtain input with identifier as data directly
            if 'in_wfs2' in request.inputs:
                wfs2 = request.inputs['in_wfs2'][0].data

                # create file, w: write in text mode
                in_path = tempfile.mkstemp(prefix='wfs2_data_', suffix='.gml')[1]
                with open(in_path, 'w') as fp:
                    fp.write(wfs2)
                    fp.close()

                LOGGER.debug('WFS2 Data String:' + str(wfs2[0:1000]))

                # open file and layer
                wfs2_src = ogr.Open(in_path)
                wfs2_lyr = wfs2_src.GetLayer()

                # get spatial reference
                wfs_epsg = int(wfs2_lyr.GetSpatialRef().GetAttrValue('AUTHORITY', 1))

                LOGGER.debug('WFS2 Feature Count in ' + str(wfs_epsg) + ':' + str(wfs2_lyr.GetFeatureCount()))

                # check spatial reference
                if wfs_epsg == self.epsg:
                    wfs2_lyr.SetSpatialFilter(geom)
                else:
                    LOGGER.debug('Incompatible Spatial Reference of WFS2 and Selection Geometry.')

                LOGGER.debug('WFS2 Feature Count After Filter:' + str(wfs2_lyr.GetFeatureCount()))

                # set output format definition
                out_path = tempfile.mkstemp(prefix='wfs_' + wfs2_lyr.GetName() + '_data_', suffix='.gml')[1]
                out_src = ogr.GetDriverByName("GML").CreateDataSource(out_path)
                out_src.CopyLayer(wfs2_lyr, wfs2_lyr.GetName())

                # set output format and file name
                response.outputs['out_wfs2'].output_format = Format(mime_type='text/xml', extension='.gml',
                                                                     schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                                                     encoding='UTF-8', validate=None)
                response.outputs['out_wfs2'].file = out_path
        else:
            # remove output from response
            del response.outputs['out_wfs2']

        # DATABASE PART ##################################################

        if 'out_db1' in request.outputs.keys():
            # check and obtain input with identifier as data directly
            if 'in_db1' in request.inputs:
```

```
                db1 = str(request.inputs['in_db1'][0].data)

                LOGGER.debug('DB1 Data Request:' + db1)

                # call spatial data export methods
                if db1 == 'poi':
                    db1_data = geolib.pg_export(db1, geom, self.epsg)
                elif db1 == 'local_plan':
                    db1_data = geolib.pg_export(db1, geom, self.epsg)
                elif db1 == 'parcel':
                    db1_data = geolib.pg_export(db1, geom, self.epsg)
                elif db1 == 'building':
                    db1_data = geolib.pg_export(db1, geom, self.epsg)
                elif db1 == 'address':
                    db1_data = geolib.pg_export(db1, geom, self.epsg)
                else:
                    db1_data = None

                # set output format and file name
                response.outputs['out_db1'].output_format = Format(mime_type='text/xml', extension='.gml',
                                                                   schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                                                   encoding='UTF-8', validate=None)
                response.outputs['out_db1'].file = db1_data
        else:
            # remove output from response
            del response.outputs['out_db1']

        if 'out_db2' in request.outputs.keys():
            # check and obtain input with identifier as data directly
            if 'in_db2' in request.inputs:
                db2 = str(request.inputs['in_db2'][0].data)

                LOGGER.debug('DB2 Data Request:' + db2)

                # call spatial data export methods
                if db2 == 'poi':
                    db2_data = geolib.pg_export(db2, geom, self.epsg)
                elif db2 == 'local_plan':
                    db2_data = geolib.pg_export(db2, geom, self.epsg)
                elif db2 == 'parcel':
                    db2_data = geolib.pg_export(db2, geom, self.epsg)
                elif db2 == 'building':
                    db2_data = geolib.pg_export(db2, geom, self.epsg)
                elif db2 == 'address':
                    db2_data = geolib.pg_export(db2, geom, self.epsg)
                else:
                    db2_data = None

                # set output format and file name
                response.outputs['out_db2'].output_format = Format(mime_type='text/xml', extension='.gml',
                                                                   schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                                                   encoding='UTF-8', validate=None)
                response.outputs['out_db2'].file = db2_data
        else:
            # remove output from response
            del response.outputs['out_db2']

        # OVERVIEW MAP AND SELECTION GEOMETRY ##################################################

        if 'out_bound' in request.outputs.keys():
            # set output format and file name
            response.outputs['out_bound'].output_format = Format(mime_type='text/xml', extension='.gml',
                                                                 schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                                                 encoding='UTF-8', validate=None)
            response.outputs['out_bound'].file = in_geom
        else:
            # remove output from response
            del response.outputs['out_bound']
```

```python

            if 'out_map' in request.outputs.keys():
                # WMS request
                url = "http://mapbender/wms7/gdm_atkis/gdm_atkis?"
                wms = WebMapService(url, version="1.1.1")

                # get extent and bounding box
                extent = geom.GetEnvelope()
                bbx1, bby1 = extent[0], extent[2]
                bbx2, bby2 = extent[1], extent[3]

                # image ratio values
                x_diff = bbx2 - bbx1
                y_diff = bby2 - bby1
                width = 1280

                # request parameters
                bbox = (bbx1, bby1, bbx2, bby2)
                size = (x_diff/y_diff*width, width)
                srs = 'EPSG:' + str(self.epsg)
                file_type = 'image/tiff'

                try:
                    # get map request
                    gm = wms.getmap(layers=['atkis1'], bbox=bbox, size=size, format=file_type, srs=srs, transparent=True)

                    LOGGER.debug('Get Map URL:' + gm.geturl())

                    # create file, wb: write in binary mode
                    ov_map_path = tempfile.mkstemp(prefix='ov_map_', suffix='.tif')[1]
                    with open(ov_map_path, 'wb') as fp:
                        fp.write(gm.read())
                        fp.close()
                except owslib.util.ServiceException as se:
                    ov_map_path = ''
                    LOGGER.debug('WMS ServiceException:' + str(se))

                # set output format and file name
                response.outputs['out_map'].output_format = Format(mime_type='image/geotiff', extension='.tif')
                response.outputs['out_map'].file = ov_map_path
            else:
                # remove output from response
                del response.outputs['out_map']
        else:
            LOGGER.debug('Only one single polygon input feature allowed. ' + str(in_lyr.GetFeatureCount()) +
                         ' features of type ' + str(geom.GetGeometryName()) + ' detected!')

        return response
```

Listing A.4: Export vector data process

A.1.5 Export 3D related spatial data process

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/proc_export_3d_data.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" The process returns 3D related spatial data selected by input geometry.
"""

# libs
import logging
```

```python
import tempfile
import requests
import owslib.util
import psycopg2
from psycopg2 import sql
from pywps import Process, ComplexInput, ComplexOutput, Format
from pywps.app.Common import Metadata
from pywps.validator.mode import MODE
from pywps.validator import complexvalidator
from owslib.wcs import WebCoverageService
from osgeo import ogr
from osgeo import osr
from lxml import etree
from lib import geolib
from lib import varlib

# authorship information
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")


# process returns 3D related spatial data selected by input geometry
class Export3dData(Process):
    # static class variables
    epsg = 25832  # local spatial reference code
    epsg3 = 31467  # outdated local spatial reference code

    def __init__(self):
        in_geom = ComplexInput(
            'in_geom',
            'Selection Geometry [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            mode=MODE.NONE
        )

        out_dem = ComplexOutput(
            'out_dem',
            'Digital Elevation Model',
            supported_formats=[Format(mime_type='image/geotiff', extension='.tif')]
        )

        out_city = ComplexOutput(
            'out_city',
            '3D City Model',
            supported_formats=[Format(mime_type='text/xml', extension='.x3d',
                                      schema='http://www.web3d.org/specifications/x3d-3.3.xsd',
                                      validate=None, encoding='UTF-8')]
        )

        inputs = [in_geom]

        outputs = [out_dem, out_city]

        super(Export3dData, self).__init__(
            self._handler,
            identifier='export_3d_data',
            version='1.0',
            title='Export 3D Related Spatial Data Process',
            abstract='The process returns 3D related spatial data selected by input geometry. Supported outputs are: '
```

```
                'Digital Elevation Model [out_dem]; 3D City Model [out_city]',
            metadata=[Metadata('The process returns 3D related spatial data selected by input geometry.',
                               'http://geodev:8080/geonetwork/srv/ger/catalog.search?service=CSW&version=2.0.2'
                               '&request=GetRecordById&id=c850b578-8561-42fb-88d1-1ac9e3314cf4#/metadata/'
                               'c850b578-8561-42fb-88d1-1ac9e3314cf4')],
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    # handler method obtains request object and response object
    # @staticmethod  # only for static methods, no 'self' applicable
    def _handler(self, request, response):
        # check if data is given by reference
        if request.inputs['in_geom'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_geom'][0].method == 'GET':
                # obtain input with identifier as file name
                in_geom = request.inputs['in_geom'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_geom'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_geom'][0].data

                LOGGER.debug('XML Response:' + in_response)

                # get content of LiteralData, Reference or ComplexData
                ref_url = varlib.get_output(etree.fromstring(in_response))

                # get GML file as reference
                r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
                data = r.content

                # create file, w: write in text mode
                filename = tempfile.mkstemp(prefix='input_', suffix='.gml')[1]
                with open(filename, 'w') as fp:
                    fp.write(data)
                    fp.close()

                in_geom = filename
        else:
            # obtain input with identifier as file name
            in_geom = request.inputs['in_geom'][0].file

        # open file and layer
        in_src = ogr.Open(in_geom)
        in_lyr = in_src.GetLayer()

        # get spatial reference
        epsg0 = int(in_lyr.GetSpatialRef().GetAttrValue('AUTHORITY', 1))

        # only one single input feature
        if in_lyr.GetFeatureCount() == 1:
            # get geometry and extent
            feat = in_lyr.GetNextFeature()
            geom = feat.GetGeometryRef()
            extent = geom.GetEnvelope()

            # harmonization of spatial reference
            if epsg0 != self.epsg:
                # transform extent to local spatial reference
                bbx1, bby1 = geolib.geo_transform(extent[0], extent[2], epsg0, self.epsg)
                bbx2, bby2 = geolib.geo_transform(extent[1], extent[3], epsg0, self.epsg)
            else:
                bbx1, bby1 = extent[0], extent[2]
                bbx2, bby2 = extent[1], extent[3]

```

```
        LOGGER.debug('Input BBox in ' + str(self.epsg) + ':' + str(bbx1) +
                     ',' + str(bby1) + ',' + str(bbx2) + ',' + str(bby2))

        # DEM PART ##################################################

        if 'out_dem' in request.outputs.keys():
            # WCS request
            url = "http://mapbender/wcs7/verma_hoehen/verma_dgm?"
            wcs = WebCoverageService(url, version="1.0.0")

            # list all coverages
            LOGGER.debug(','.join(wcs.contents))

            # get a certain coverage
            dem = wcs['dgm1']

            # list all attributes of the coverage
            LOGGER.debug(dir(dem))

            # list all bbox
            for bb in dem.boundingboxes:
                LOGGER.debug('DEM BBox:' + str(bb) + '_' +
                             str(dem.boundingboxes[1]['nativeSrs']) + '_' +
                             str(dem.boundingboxes[1]['bbox']))

            # list all time positions
            for tp in dem.timepositions:
                LOGGER.debug('DEM TPos:' + str(tp))

            # list all supported formats
            for sf in dem.supportedFormats:
                LOGGER.debug('DEM Formats:' + str(sf))

            # request parameters
            bbox = (bbx1, bby1, bbx2, bby2)
            crs = 'EPSG:' + str(self.epsg)
            file_type = 'GEOTIFF_16'  # GEOTIFF_16, AAIGRID, GTiff
            resx, resy = 1, 1  # max. available resolution of DEM data

            try:
                # get coverage request
                gc = wcs.getCoverage(identifier=dem.id, bbox=bbox, format=file_type, crs=crs, resx=resx, resy=resy)

                LOGGER.debug('Get Coverage URL:' + gc.geturl())

                # create file, wb: write in binary mode
                dem_path = tempfile.mkstemp(prefix='dem_', suffix='.tif')[1]
                with open(dem_path, 'wb') as fp:
                    fp.write(gc.read())
                    fp.close()
            except owslib.util.ServiceException as se:
                dem_path = ''
                LOGGER.debug('WCS ServiceException:' + str(se))

            # set output format and file name
            response.outputs['out_dem'].output_format = Format(mime_type='image/geotiff', extension='.tif')
            response.outputs['out_dem'].file = dem_path
        else:
            # remove output from response
            del response.outputs['out_dem']

        # 3D CITY MODEL PART ##################################################

        if 'out_city' in request.outputs.keys():
            # harmonization of spatial reference
            if epsg0 != self.epsg3:
                # transform selection geometry to spatial reference of 3D city model
                sref0 = osr.SpatialReference()
```

```
                    sref0.ImportFromEPSG(epsg0)
                    sref3 = osr.SpatialReference()
                    sref3.ImportFromEPSG(self.epsg3)
                    transform = osr.CoordinateTransformation(sref0, sref3)
                    geom.Transform(transform)

                LOGGER.debug('Input Geometry in ' + str(self.epsg3) + ':' + geom.ExportToWkt())

                # open database connection, using .pgpass for authentication
                db_conn = psycopg2.connect("host=geodb port=5432 dbname=citydb_v4 user=postgres")

                # check connection
                if db_conn is None:
                    LOGGER.debug('PG connection refused.')

                # open cursor to perform database operations
                db_cur = db_conn.cursor()

                # sql query with placeholders, transformation to local spatial reference
                query = sql.SQL("SELECT ST_AsX3D(ST_Transform(ST_SetSRID(sg.geometry, %s), %s), 3, 0) AS geom_3d "
                                "FROM {tbl} sg LEFT JOIN thematic_surface ts ON ts.lod2_multi_surface_id = sg.root_id "
                                "LEFT JOIN building b ON ts.building_id = b.building_root_id "
                                "WHERE sg.geometry IS NOT NULL AND ts.lod2_multi_surface_id IS NOT NULL "
                                "AND ST_Intersects(ST_SetSRID(ST_PolygonFromText(%s), %s), sg.geometry);")

                # execute command, using templating mechanism for better security
                db_cur.execute(query.format(tbl=sql.Identifier('surface_geometry')),
                               [self.epsg3, self.epsg, geom.ExportToWkt(), self.epsg3])

                # process query result data
                city_data = '<?xml version="1.0" encoding="UTF-8"?>\n' \
                            '<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.3//EN"\n' \
                            '  "http://www.web3d.org/specifications/x3d-3.3.dtd">\n\n' \
                            '<X3D profile="Interchange" version="3.3"\n' \
                            '     xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"\n' \
                            '     xsd:noNamespaceSchemaLocation="http://www.web3d.org/specifications/x3d-3.3.xsd">\n' \
                            '<Scene>'

                for city_geom in db_cur:
                    city_data += '\n  <Shape>\n    ' + str(city_geom)[2:-3] + '\n  </Shape>'

                city_data += '\n</Scene>\n</X3D>'

                # create file, w: write in text mode
                city_path = tempfile.mkstemp(prefix='city_', suffix='.x3d')[1]
                with open(city_path, 'w') as fp:
                    fp.write(city_data)
                    fp.close()

                # make the changes to the database persistent
                db_conn.commit()

                # close communication with the database
                db_cur.close()
                db_conn.close()

                # free and reassign
                db_conn = None

                # set output format and file name
                response.outputs['out_city'].output_format = Format(mime_type='text/xml', extension='.x3d',
                                                                    schema='http://www.web3d.org/specifications/x3d-3.3.xsd',
                                                                    validate=None, encoding='UTF-8')
                response.outputs['out_city'].file = city_path
            else:
                # remove output from response
                del response.outputs['out_city']
        else:
```

```
            LOGGER.debug('Only one single input feature allowed. ' +
                         str(in_lyr.GetFeatureCount()) + ' detected!')

        LOGGER.debug(request.outputs.keys())
        LOGGER.debug(response.outputs.keys())

        return response
```

Listing A.5: Export 3D related spatial data process

A.1.6 APOLLO rough danger distance process

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/proc_apollo_rough_dist.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" The process is part of the explosive ordnance disposal workflow
    and returns rough danger distance based on given solid and tnt mass.
"""

# libs
import logging
from pywps import Process, LiteralInput, LiteralOutput
from pywps.app.Common import Metadata
from pywps.validator.allowed_value import RANGECLOSURETYPE, ALLOWEDVALUETYPE
from pywps.inout.literaltypes import AllowedValue
from lib import geolib

# authorship information
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")


# process returns rough danger distance based on given solid and tnt mass
class ApolloRoughDist(Process):
    def __init__(self):
        in_tnt = LiteralInput(
            'in_tnt',
            'Rough TNT Blast Power [kg]',
            data_type='integer',
            # spacing unable to use due incompatibilities between QGIS wps client
            # allowed_values=(range(50, 2000+1, 50)),
            allowed_values=[AllowedValue(minval=1, maxval=5000,  # spacing=50,
                                         allowed_type=ALLOWEDVALUETYPE.RANGE,
                                         range_closure=RANGECLOSURETYPE.OPEN)]
        )

        in_solid = LiteralInput(
            'in_solid',
            'Solid Type',
            abstract='Type of material the damage distance threshold will be calculated for: '
                     '0 = Float Glass, 1 = Eardrum Rupture',
            data_type='integer',
            allowed_values=(0, 1),
            min_occurs=0
```

```
        )

        out_rough_dist = LiteralOutput(
            'out_rough_dist',
            'Rough Danger Distance',
            data_type='string'  # use of string instead float as workaround for bug in PyWPS
        )

        inputs = [in_tnt, in_solid]

        outputs = [out_rough_dist]

        super(ApolloRoughDist, self).__init__(
            self._handler,
            identifier='apollo_rough_dist',
            version='1.0',
            title='APOLLO Rough Danger Distance Process',
            abstract='The process returns rough danger distance based on given solid and tnt mass.',
            metadata=[Metadata('The process is part of the explosive ordnance disposal workflow '
                               'and returns rough danger distance based on given solid and tnt mass.',
                               'http://geodev:8080/geonetwork/srv/ger/catalog.search?service=CSW&version=2.0.2'
                               '&request=GetRecordById&id=c850b578-8561-42fb-88d1-1ac9e3314cf4#/metadata/'
                               'c850b578-8561-42fb-88d1-1ac9e3314cf4')],
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    # handler method obtains request object and response object
    # @staticmethod  # only for static methods, no 'self' applicable
    def _handler(self, request, response):
        # default parameter values
        tnt, solid = 0, 0

        # check and obtain input with identifier as data directly
        if 'in_tnt' in request.inputs:
            tnt = request.inputs['in_tnt'][0].data
        if 'in_solid' in request.inputs:
            solid = request.inputs['in_solid'][0].data

        # calculation of threshold distance
        dist_threshold = geolib.damage_dist_threshold(tnt, solid)

        # set output format and file name
        response.outputs['out_rough_dist'].data = str(dist_threshold)

        return response
```

Listing A.6: APOLLO rough danger distance process

A.1.7 APOLLO configuration process

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/proc_apollo_conf.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" The process is part of the explosive ordnance disposal workflow
    and returns APOLLO configuration data for SIRIUS interface.
"""

# libs
```

```
import logging
import tempfile
import json
from pywps import Process, LiteralInput, ComplexInput, ComplexOutput, Format
from pywps.app.Common import Metadata
from pywps.validator.mode import MODE
from pywps.validator import complexvalidator
from pywps.validator.allowed_value import RANGECLOSURETYPE, ALLOWEDVALUETYPE
from pywps.inout.literaltypes import AllowedValue
from easydict import EasyDict
from osgeo import ogr
from lib import geolib

# authorship information
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")


# process returns APOLLO configuration data for SIRIUS interface
class ApolloConf(Process):
    # static class variables
    epsg = 25832  # local spatial reference code
    epsg2 = 4326  # spatial reference code for WGS84
    srv_url = 'https://www.cadfem.de/apollo/'  # url provided by the SIRIUS project team

    def __init__(self):
        in_geom = ComplexInput(
            'in_geom',
            'Exact Location [gml]',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      validate=complexvalidator.validategml)],
            # validation mode unable to use due incompatibilities between mimetype library and QGIS wps client
            mode=MODE.NONE
        )

        in_precision = LiteralInput(
            'in_precision',
            'Precision [m]',
            abstract='Precision used by APOLLO simulation. Supported values are: 0.5, 1.0, 2.5, 5.0, 10.0',
            data_type='float',
            allowed_values=(0.5, 1.0, 2.5, 5.0, 10.0)
        )

        in_height = LiteralInput(
            'in_height',
            'Relative Height [m]',
            data_type='float',
            default='-2.5'
        )

        in_tnt = LiteralInput(
            'in_tnt',
            'Exact TNT Blast Power [kg]',
            data_type='integer',
            # spacing unable to use due incompatibilities between QGIS wps client
            # allowed_values=(range(50, 2000+1, 50)),
            allowed_values=[AllowedValue(minval=1, maxval=5000,  # spacing=50,
                                         allowed_type=ALLOWEDVALUETYPE.RANGE,
                                         range_closure=RANGECLOSURETYPE.OPEN)]
        )
```

```python

        in_heading = LiteralInput(
            'in_heading',
            'Bomb Azimuth Angle [deg]',
            data_type='float',
            min_occurs=0,
            default='0.0'
        )

        in_pitch = LiteralInput(
            'in_pitch',
            'Bomb Tilt Angle [deg]',
            data_type='float',
            min_occurs=0,
            default='0.0'
        )

        in_type = LiteralInput(
            'in_type',
            'Bomb Type',
            abstract='Type of the bomb after classification. Supported values are: N/A, GP100, GP250',
            data_type='string',
            allowed_values=('N/A', 'GP100', 'GP250'),
            min_occurs=0
        )

        in_detonator = LiteralInput(
            'in_detonator',
            'Detonator Position',
            abstract='Position of detonator after classification. Supported values are: N/A, Front, Rear, Top, Bottom',
            data_type='string',
            allowed_values=('N/A', 'Front', 'Rear', 'Top', 'Bottom'),
            min_occurs=0
        )

        in_site_desc = LiteralInput(
            'in_site_desc',
            'Site Description',
            abstract='Description of the bomb find location. Supported values are: Surface, Cavern',
            data_type='string',
            allowed_values=('Surface', 'Cavern'),
            min_occurs=0
        )

        in_site_rad = LiteralInput(
            'in_site_rad',
            'Site Radius [m]',
            data_type='float',
            min_occurs=0,
            default='0.0'
        )

        in_hidden = LiteralInput(
            'in_hidden',
            'Hidden Objects [gml:id1 gml:id2]',
            abstract='List of 3D city model objects that will be ignored by the simulation. '
                     'Supported values are GML identification strings.',
            data_type='string',
            min_occurs=0
        )

        out_conf = ComplexOutput(
            'out_conf',
            'APOLLO Configuration Data',
            supported_formats=[Format(mime_type='application/json', extension='.json',
                                      validate=complexvalidator.validategeojson,
                                      encoding='UTF-8', schema='json')]
        )
```

```
        inputs = [in_geom, in_precision, in_height, in_tnt, in_heading, in_pitch,
                  in_type, in_detonator, in_site_desc, in_site_rad, in_hidden]

        outputs = [out_conf]

        super(ApolloConf, self).__init__(
            self._handler,
            identifier='apollo_conf',
            version='1.0',
            title='APOLLO Configuration Process',
            abstract='The process returns APOLLO configuration data for SIRIUS interface.',
            metadata=[Metadata('The process is part of the explosive ordnance disposal workflow '
                               'and returns APOLLO configuration data for SIRIUS interface.',
                               'http://geodev:8080/geonetwork/srv/ger/catalog.search?service=CSW&version=2.0.2'
                               '&request=GetRecordById&id=c850b578-8561-42fb-88d1-1ac9e3314cf4#/metadata/'
                               'c850b578-8561-42fb-88d1-1ac9e3314cf4')],
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    # handler method obtains request object and response object
    # @staticmethod  # only for static methods, no 'self' applicable
    def _handler(self, request, response):
        # obtain input with identifier as file name
        in_file = request.inputs['in_geom'][0].file

        # possible request attributes: 'abstract', 'as_reference', 'base64', 'clone', 'crs', 'crss', 'data',
        # 'describe_xml', 'dimensions', 'execute_xml', 'file', 'get_base64', 'get_data', 'get_file',
        # 'get_memory_object', 'get_stream', 'get_workdir', 'identifier', 'json', 'll', 'max_occurs', 'memory_object',
        # 'metadata', 'min_occurs', 'set_base64', 'set_data', 'set_file', 'set_memory_object', 'set_stream',
        # 'set_workdir', 'source', 'source_type', 'stream', 'title', 'ur', 'valid_mode', 'validator', 'workdir'

        # default parameter values
        bomb_type, detonator, site_desc, hidden = '', '', '', ''
        precision, height, tnt, heading, pitch, site_rad = 0., 0., 0, 0., 0., 2.

        # check and obtain input with identifier as data directly
        if 'in_precision' in request.inputs:
            precision = request.inputs['in_precision'][0].data
        if 'in_height' in request.inputs:
            height = request.inputs['in_height'][0].data
        if 'in_tnt' in request.inputs:
            tnt = request.inputs['in_tnt'][0].data
        if 'in_heading' in request.inputs:
            heading = request.inputs['in_heading'][0].data
        if 'in_pitch' in request.inputs:
            pitch = request.inputs['in_pitch'][0].data
        if 'in_type' in request.inputs:
            bomb_type = request.inputs['in_type'][0].data
        if 'in_detonator' in request.inputs:
            detonator = request.inputs['in_detonator'][0].data
        if 'in_site_desc' in request.inputs:
            site_desc = request.inputs['in_site_desc'][0].data
        if 'in_site_rad' in request.inputs:
            site_rad = request.inputs['in_site_rad'][0].data
        if 'in_hidden' in request.inputs:
            hidden = (request.inputs['in_hidden'][0].data).split()

        # open file and layer
        in_src = ogr.Open(in_file)
        in_lyr = in_src.GetLayer()

        # only one single input feature and valid tnt blast power
        if in_lyr.GetFeatureCount() == 1 and tnt > 0:
            # conservative calculation for float glass
```

```
            dist_threshold = geolib.damage_dist_threshold(tnt, 0)

            LOGGER.debug('Threshold:' + str(dist_threshold))

            # get the feature geometry
            in_feat = in_lyr.GetNextFeature()
            in_geom = in_feat.GetGeometryRef()

            # get SRID of geometry and make sure location is a point
            epsg0 = int(in_geom.GetSpatialReference().GetAttrValue('AUTHORITY', 1))
            x0, y0 = in_geom.Centroid().GetX(), in_geom.Centroid().GetY()

            # harmonization of spatial reference
            if epsg0 != self.epsg:
                # transform position to local spatial reference
                x2, y2 = geolib.geo_transform(x0, y0, epsg0, self.epsg)
            else:
                x2, y2 = x0, y0

            # calculate bounding box
            bbx1 = x2 - dist_threshold
            bby1 = y2 - dist_threshold
            bbx2 = x2 + dist_threshold
            bby2 = y2 + dist_threshold

            # transform position to WGS84
            x_wgs, y_wgs = geolib.geo_transform(x2, y2, self.epsg, self.epsg2)

            LOGGER.debug('Coordinates in ' + str(self.epsg) + ':' + str(x2) + '/' + str(y2))
            LOGGER.debug('Coordinates in ' + str(self.epsg2) + ':' + str(x_wgs) + '/' + str(y_wgs))

            # create location geometry
            location = ogr.Geometry(ogr.wkbPoint)
            location.AddPoint(x2, y2)

            LOGGER.debug('Location as WKT:' + location.ExportToWkt())

            # create output data
            conf_data = EasyDict({'bomb': {'tnt': tnt, 'type': bomb_type, 'detonator': detonator},
                                  'domain': {'name': 'Ultimo', 'zroi': 100, 'droi': dist_threshold},
                                  'mode': {'name': 'Ultimo', 't': 50, 'precision': precision},
                                  'site': {'type': site_desc, 'radius': site_rad},
                                  'geometry': {'crs': self.epsg2, 'position': [x_wgs, y_wgs], 'depth': (-1) * height},
                                  'crs': self.epsg,
                                  'position': [x2, y2],
                                  'height': height,
                                  'heading': heading,
                                  'pitch': pitch,
                                  'extent': [bbx1, bby1, bbx2, bby2],
                                  'hiddenObjects': hidden,
                                  'service': {'url': self.srv_url, 'resultFile': 'effects_' + str(self.uuid) + '.zip'}
                                  })

            conf_json = json.dumps(conf_data)

            # create file, w: write in text mode
            conf_path = tempfile.mkstemp(prefix='conf_', suffix='.json')[1]
            with open(conf_path, 'w') as fp:
                fp.write(conf_json)
                fp.close()

            # set output format and file name
            response.outputs['out_conf'].output_format = Format(mime_type='application/json', extension='.json',
                                                                validate=complexvalidator.validategeojson,
                                                                encoding='UTF-8', schema='json')
            response.outputs['out_conf'].file = conf_path
        else:
            # remove output from response
```

```
            del response.outputs['out_conf']

            LOGGER.debug('Only one single input feature allowed. ' +
                         str(in_lyr.GetFeatureCount()) + ' detected!')

        # free and reassign
        in_src = None
        in_lyr = None

        # possible response attributes: 'abstract', 'as_reference', 'base64', 'crs', 'crss', 'data', 'describe_xml',
        # 'dimensions', 'execute_xml', 'file', 'get_base64', 'get_data', 'get_file', 'get_memory_object', 'get_stream',
        # 'get_workdir', 'identifier', 'json', 'll', 'max_occurs', 'memory_object', 'metadata', 'min_occurs',
        # 'set_base64', 'set_data', 'set_file', 'set_memory_object', 'set_stream', 'set_workdir', 'source',
        # 'source_type', 'stream', 'title', 'ur', 'valid_mode', 'validator', 'workdir'

        return response
```

Listing A.7: APOLLO configuration process

A.1.8 APOLLO execute process

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/proc_apollo_execute.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" The process is part of the explosive ordnance disposal workflow
    and executes APOLLO via SIRIUS and returns blast effects result.
"""

# libs
import logging
import tempfile
import requests
import json
from pywps import Process, ComplexInput, ComplexOutput, Format
from pywps.app.Common import Metadata
from pywps.validator.mode import MODE
from pywps.validator import complexvalidator
from lxml import etree
from lib import varlib

# authorship information
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")


# process executes APOLLO via SIRIUS and returns blast effects result
class ApolloExecute(Process):
    def __init__(self):
        in_conf = ComplexInput(
            'in_conf',
            'APOLLO Configuration Data [json]',
            supported_formats=[Format(mime_type='application/json', extension='.json',
                                      validate=complexvalidator.validategeojson,
                                      encoding='UTF-8', schema='json')],
```

```
            mode=MODE.NONE
        )

        in_dem = ComplexInput(
            'in_dem',
            'Digital Elevation Model [tif]',
            supported_formats=[Format(mime_type='image/geotiff', extension='.tif')],
            mode=MODE.NONE
        )

        in_city = ComplexInput(
            'in_city',
            '3D City Model [x3d]',
            supported_formats=[Format(mime_type='text/xml', extension='.x3d',
                                      schema='http://www.web3d.org/specifications/x3d-3.3.xsd',
                                      validate=None, encoding='UTF-8')],
            mode=MODE.NONE
        )

        out_effects = ComplexOutput(
            'out_effects',
            'APOLLO Effects Result',
            supported_formats=[Format(mime_type='application/octet-stream')]
        )

        inputs = [in_conf, in_dem, in_city]

        outputs = [out_effects]

        super(ApolloExecute, self).__init__(
            self._handler,
            identifier='apollo_execute',
            version='1.0',
            title='APOLLO Execute Process',
            abstract='The process executes APOLLO via SIRIUS and returns blast effects result.',
            metadata=[Metadata('The process is part of the explosive ordnance disposal workflow '
                               'and executes APOLLO via SIRIUS and returns blast effects result.',
                               'http://geodev:8080/geonetwork/srv/ger/catalog.search?service=CSW&version=2.0.2'
                               '&request=GetRecordById&id=c850b578-8561-42fb-88d1-1ac9e3314cf4#/metadata/'
                               'c850b578-8561-42fb-88d1-1ac9e3314cf4')],
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    # handler method obtains request object and response object
    # @staticmethod  # only for static methods, no 'self' applicable
    def _handler(self, request, response):
        # IN_CONF PART ##################################################

        # check if data is given by reference
        if request.inputs['in_conf'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_conf'][0].method == 'GET':
                # obtain input with identifier as file name
                in_conf = request.inputs['in_conf'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_conf'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_conf'][0].data

                LOGGER.debug('XML Response:' + in_response)

                # get content of LiteralData, Reference or ComplexData
                ref_url = varlib.get_output(etree.fromstring(in_response))

                # get GML file as reference
```

```
                r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
                data = r.content

                # create file, w: write in text mode
                filename = tempfile.mkstemp(prefix='conf_', suffix='.json')[1]
                with open(filename, 'w') as fp:
                    fp.write(data)
                    fp.close()

                in_conf = filename
        else:
            # obtain input with identifier as file name
            in_conf = request.inputs['in_conf'][0].file

        # IN_DEM PART ##################################################

        # check if data is given by reference
        if request.inputs['in_dem'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_dem'][0].method == 'GET':
                # obtain input with identifier as file name
                in_dem = request.inputs['in_dem'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_dem'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_dem'][0].data

                LOGGER.debug('XML Response:' + in_response)

                # get content of LiteralData, Reference or ComplexData
                ref_url = varlib.get_output(etree.fromstring(in_response))

                # get GML file as reference
                r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
                data = r.content

                # create file, wb: write in binary mode
                filename = tempfile.mkstemp(prefix='dem_', suffix='.tif')[1]
                with open(filename, 'wb') as fp:
                    fp.write(data)
                    fp.close()

                in_dem = filename
        else:
            # obtain input with identifier as file name
            in_dem = request.inputs['in_dem'][0].file

        # IN_CITY PART ##################################################

        # check if data is given by reference
        if request.inputs['in_city'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_city'][0].method == 'GET':
                # obtain input with identifier as file name
                in_city = request.inputs['in_city'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_city'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_city'][0].data

                LOGGER.debug('XML Response:' + in_response)

                # get content of LiteralData, Reference or ComplexData
                ref_url = varlib.get_output(etree.fromstring(in_response))

                # get GML file as reference
                r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
                data = r.content
```

```
                # create file, w: write in text mode
                filename = tempfile.mkstemp(prefix='city_', suffix='.x3d')[1]
                with open(filename, 'w') as fp:
                    fp.write(data)
                    fp.close()

                in_city = filename
        else:
            # obtain input with identifier as file name
            in_city = request.inputs['in_city'][0].file

        # EXECUTE PART ##################################################

        LOGGER.debug('Config path:' + in_conf)
        LOGGER.debug('DEM path:' + in_dem)
        LOGGER.debug('City path:' + in_city)

        # open configuration file
        with open(in_conf, 'r') as fp:
            conf_data = json.load(fp)

        # read url for APOLLO service and result data
        if 'service' in conf_data:
            srv_url = conf_data['service']['url']
            result_file = conf_data['service']['resultFile']
            srv_url_result = srv_url + result_file
            LOGGER.debug('Service URL:' + srv_url_result)

        # NON-PRODUCTIVE ONLY -> overwrite result data url because simulation of working SIRIUS / APOLLO server
        srv_url_result = 'https://geodev2/apollo_result/apollo_effects.zip'

        # reveal input data, execute APOLLO and calculate effects result
        # r_exe = requests.get(srv_url, verify=False)

        # effects result file checker
        while not requests.head(srv_url_result, verify=False).status_code == requests.codes.ok:
            response.update_status('APOLLO Execute Process Still In Progress', 0)
            LOGGER.debug('Resource File Status Code:' + str(requests.head(srv_url_result, verify=False).status_code))

        # get effects result file when APOLLO is ready
        r = requests.get(srv_url_result, verify=False)
        data = r.content

        # create file, wb: write in binary mode
        result_file = tempfile.mkstemp(prefix='effects_', suffix='.zip')[1]
        with open(result_file, 'wb') as fp:
            fp.write(data)
            fp.close()

        # set output format and file name
        response.outputs['out_effects'].output_format = Format(mime_type='application/octet-stream')
        response.outputs['out_effects'].file = result_file

        return response
```

Listing A.8: APOLLO execute process

A.1.9 APOLLO evacuation zone process

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/proc_apollo_evac_zone.py

```
#!/usr/bin/env python
```

```python
# -*- coding: utf-8 -*-

""" The process is part of the explosive ordnance disposal workflow
    and returns evacuation zone around blast affected area.
"""

# libs
import numpy as np
import logging
import tempfile
import requests
import math
import os
import json
import zipfile
import shutil
from pywps import Process, LiteralInput, ComplexInput, ComplexOutput, Format
from pywps.app.Common import Metadata
from pywps.validator.mode import MODE
from pywps.validator import complexvalidator
from osgeo import ogr
from osgeo import osr
from osgeo import gdal
from lxml import etree
from lib import varlib

# authorship information
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")


# process returns evacuation zone around blast affected area
class ApolloEvacZone(Process):
    # static class variables
    rot_deg = 28.5  # z axis rotation used by APOLLO, case study only, will be 0.0 in productive use

    def __init__(self):
        in_conf = ComplexInput(
            'in_conf',
            'APOLLO Configuration Data [json]',
            supported_formats=[Format(mime_type='application/json', extension='.json',
                                      validate=complexvalidator.validategeojson,
                                      encoding='UTF-8', schema='json')],
            mode=MODE.NONE
        )

        in_effects = ComplexInput(
            'in_effects',
            'APOLLO Effects Result [zip|dat]',
            supported_formats=[Format(mime_type='application/octet-stream', extension='.zip')],
            mode=MODE.NONE
        )

        in_dmg_lvl = LiteralInput(
            'in_dmg_lvl',
            'Damage Level',
            abstract='Level of damage the evacuation zone will be calculated for: '
                     '0 = Float Glass, '
                     '1 = Hardened Glass, '
                     '2 = Safety Glass, '
                     '3 = Masonry, '
```

```
                      '4 = Eardrum Rupture, '
                      '5 = Injury, '
                      '6 = Lethal Injury',
            data_type='integer',
            allowed_values=(0, 1, 2, 3, 4, 5, 6),
            min_occurs=0
        )

        out_evac_zone = ComplexOutput(
            'out_evac_zone',
            'Evacuation Zone',
            supported_formats=[Format(mime_type='text/xml', extension='.gml',
                                      schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                      encoding='UTF-8', validate=None)]
        )

        out_raster = ComplexOutput(
            'out_raster',
            'Evacuation Raster',
            supported_formats=[Format(mime_type='image/geotiff', extension='.tif')]
        )

        inputs = [in_conf, in_effects, in_dmg_lvl]

        outputs = [out_evac_zone, out_raster]

        super(ApolloEvacZone, self).__init__(
            self._handler,
            identifier='apollo_evac_zone',
            version='1.0',
            title='APOLLO Evacuation Zone Process',
            abstract='The process returns evacuation zone around blast affected area.',
            metadata=[Metadata('The process is part of the explosive ordnance disposal workflow '
                               'and returns evacuation zone around blast affected area.',
                               'http://geodev:8080/geonetwork/srv/ger/catalog.search?service=CSW&version=2.0.2'
                               '&request=GetRecordById&id=c850b578-8561-42fb-88d1-1ac9e3314cf4#/metadata/'
                               'c850b578-8561-42fb-88d1-1ac9e3314cf4')],
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    # handler method obtains request object and response object
    # @staticmethod  # only for static methods, no 'self' applicable
    def _handler(self, request, response):
        # IN_CONF PART ##################################################

        # check if data is given by reference
        if request.inputs['in_conf'][0].as_reference:
            # check if GET method is used
            if request.inputs['in_conf'][0].method == 'GET':
                # obtain input with identifier as file name
                in_conf = request.inputs['in_conf'][0].file
            # check if POST method is used - whole response has to be parsed (chaining)
            elif request.inputs['in_conf'][0].method == 'POST':
                # obtain whole response XML with identifier as data directly
                in_response = request.inputs['in_conf'][0].data

                LOGGER.debug('XML Response:' + in_response)

                # get content of LiteralData, Reference or ComplexData
                ref_url = varlib.get_output(etree.fromstring(in_response))

                # get GML file as reference
                r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
                data = r.content

```

```
            # create file, w: write in text mode
            filename = tempfile.mkstemp(prefix='conf_', suffix='.json')[1]
            with open(filename, 'w') as fp:
                fp.write(data)
                fp.close()

            in_conf = filename
    else:
        # obtain input with identifier as file name
        in_conf = request.inputs['in_conf'][0].file

    # IN_EFFECTS PART ##################################################

    # check if data is given by reference
    if request.inputs['in_effects'][0].as_reference:
        # check if GET method is used
        if request.inputs['in_effects'][0].method == 'GET':
            # obtain input with identifier as file name
            in_effects = request.inputs['in_effects'][0].file
        # check if POST method is used - whole response has to be parsed (chaining)
        elif request.inputs['in_effects'][0].method == 'POST':
            # obtain whole response XML with identifier as data directly
            in_response = request.inputs['in_effects'][0].data

            LOGGER.debug('XML Response:' + in_response)

            # get content of LiteralData, Reference or ComplexData
            ref_url = varlib.get_output(etree.fromstring(in_response))

            # get GML file as reference
            r = requests.get(ref_url[ref_url.keys()[0]], verify=False)
            data = r.content

            # create file, wb: write in binary mode
            filename = tempfile.mkstemp(prefix='effects_', suffix='.zip')[1]
            with open(filename, 'wb') as fp:
                fp.write(data)
                fp.close()

            in_effects = filename
    else:
        # obtain input with identifier as file name
        in_effects = request.inputs['in_effects'][0].file

    # IN DAMAGE LEVEL PART ##################################################

    dmg_lvl = 'F4_FloatGl'  # default level of damage

    # check and obtain input with identifier as data directly
    if 'in_dmg_lvl' in request.inputs:
        lvl = request.inputs['in_dmg_lvl'][0].data

        if lvl == 1:
            dmg_lvl = 'F5_HardGl'
        if lvl == 2:
            dmg_lvl = 'F6_SafeGl'
        if lvl == 3:
            dmg_lvl = 'F7_Masonry'
        if lvl == 4:
            dmg_lvl = 'F10_Eardrum'
        if lvl == 5:
            dmg_lvl = 'F11_Injury'
        if lvl == 6:
            dmg_lvl = 'F12_Lethal'

    # CONFIG PART ##################################################

    LOGGER.debug('Config path:' + in_conf)
```

```
        LOGGER.debug('Effects path:' + in_effects)

        # open configuration file
        with open(in_conf, 'r') as fp:
            conf_data = json.load(fp)

        # check and obtain input with identifier as data directly
        if 'crs' in conf_data:
            epsg = conf_data['crs']
        if 'position' in conf_data:
            if len(conf_data['position']) > 1:
                x = conf_data['position'][0]
                y = conf_data['position'][1]
        if 'mode' in conf_data:
            if 'precision' in conf_data['mode']:
                precision = conf_data['mode']['precision']

        # check necessary parameter
        try:
            x, y, epsg, precision
            LOGGER.debug('Parameter:' + str(x) + '/' + str(y) + '/' + str(epsg) + '/' + str(precision))
        except NameError:
            LOGGER.debug('Input value error in APOLLO configuration.')

        # APOLLO EFFECTS PART ##################################################

        in_effects_dat = in_effects

        # zip archive handling for APOLLO effects file
        if zipfile.is_zipfile(in_effects):
            with zipfile.ZipFile(in_effects) as my_zip:
                # get name of files with *.dat extension
                cont_match = filter(lambda s: '.dat' in s, my_zip.namelist())

                # set new name for APOLLO effects file
                in_effects_dat = os.path.join(os.path.dirname(in_effects), cont_match[0])

                # extract first *.dat file
                with my_zip.open(cont_match[0]) as zf, open(in_effects_dat, 'wb') as f:
                    shutil.copyfileobj(zf, f)

        LOGGER.debug('APOLLO effects file:' + in_effects_dat)

        # build dtype array structure for APOLLO effects file
        dt = np.dtype({'names': ['I', 'J', 'K', 'Dir', 'N', 'Obj',
                                 'F1_MaxOP', 'F2_MaxOP-Imp', 'F3_OP-Imp', 'F4_FloatGl', 'F5_HardGl', 'F6_SafeGl',
                                 'F7_Masonry', 'F8_RC30-01', 'F9_RC30-06', 'F10_Eardrum', 'F11_Injury', 'F12_Lethal'],
                       'formats': ['int', 'int', 'int', 'int', 'int', 'int', 'float', 'float', 'float', 'float',
                                   'float', 'float', 'float', 'float', 'float', 'float', 'float', 'float']})

        # read APOLLO effects file
        data = np.loadtxt(in_effects_dat, skiprows=19, dtype=dt, ndmin=2)

        # get dimensions (I=512 J=512 K=76)
        size_i = np.amax(data['I']) - np.amin(data['I']) + 1
        size_j = np.amax(data['J']) - np.amin(data['J']) + 1
        # size_k = np.amax(data['K']) - np.amin(data['K']) + 1

        # get delta of translation to positive quarter
        delta_i = abs(np.amin(data['I']))
        # delta_j = abs(np.amin(data['J']))
        # delta_k = abs(np.amin(data['K']))

        # max values, no abs, needed for iterations
        # max_i = np.amax(data['I'])
        max_j = np.amax(data['J'])
        # max_k = np.amax(data['K'])
```

```
        LOGGER.debug('Dimensions:sizeI=' + str(size_i) + '/sizeJ=' + str(size_j) +
                     '/deltaI=' + str(delta_i) + '/maxJ=' + str(max_j))

        # empty array with size of ground surface
        target = np.zeros((size_j, size_i))

        # make data flat
        for row in np.nditer(data):
            # save value only if greater than previous value in K direction
            if row[dmg_lvl] > target[max_j - row['J']][delta_i + row['I']]:
                # save n-dimensional values
                # target[max_j - row['J']][delta_i + row['I']] = [row['F1_MaxOP'], row['F2_MaxOP-Imp'],
                #                                                  row['F3_OP-Imp'], row[dmg_lvl]]
                # save 1-dimensional value
                target[max_j - row['J']][delta_i + row['I']] = row[dmg_lvl]

        # free and reassign
        data = None

        # RASTER PART ##################################################

        # file path for raster
        raster_path = os.path.splitext(in_effects_dat)[0] + '_' + dmg_lvl.lower() + '_.tif'

        # set spatial reference and export projection to wkt
        sref = osr.SpatialReference()
        sref.ImportFromEPSG(epsg)
        wkt_proj = sref.ExportToWkt()

        # number of pixels in x and y, and size of one pixel
        pixel_x = size_i
        pixel_y = size_j
        pixel_size = precision

        # transform location coordinates to upper left base point used in GTiff
        rot_rad = math.radians(-1 * self.rot_deg)
        size_i2 = size_i / 2.0
        size_j2 = size_j / 2.0
        delta_x = (size_i2 * precision) * math.cos(rot_rad) + (size_j2 * precision) * math.sin(rot_rad)
        delta_y = -(size_i2 * precision) * math.sin(rot_rad) + (size_j2 * precision) * math.cos(rot_rad)
        min_x = x - delta_x
        max_y = y + delta_y

        LOGGER.debug('Coordinates:' + str(min_x) + '/' + str(max_y) + '/' + str(delta_x) + '/' + str(delta_y))
        LOGGER.debug('Rotation:' + str(math.cos(rot_rad) * pixel_size) + '/' + str(math.sin(rot_rad)))

        # set raster format definition
        raster = gdal.GetDriverByName('GTiff').Create(
            raster_path,  # file path
            pixel_x,  # width in pixels
            pixel_y,  # height in pixels
            1,  # number of bands
            gdal.GDT_Float32  # type of raster
        )

        # set transformation from pixel to projected coordinates
        raster.SetGeoTransform((
            min_x,  # x value at top left
            math.cos(rot_rad) * pixel_size,  # transform pixel size in west-east
            math.sin(rot_rad),  # rotation factor 1
            max_y,  # y value at top left
            math.sin(rot_rad),  # rotation factor 2
            -math.cos(rot_rad) * pixel_size  # transform pixel size in north-south
        ))

        # set projection for transformed coordinates
        raster.SetProjection(wkt_proj)

```

```
        # write simulated data to band 1
        raster.GetRasterBand(1).WriteArray(target)

        # flush all write cached data to disk
        raster.FlushCache()

        # free and reassign
        raster = None
        target = None

        # RASTER MASK PART ##################################################

        # file path for raster mask
        raster_mask_path = os.path.splitext(in_effects_dat)[0] + '_' + dmg_lvl.lower() + '_mask_.tif'

        # import raster
        ds_r = gdal.Open(raster_path)
        ds_r_val = ds_r.ReadAsArray()

        # spatial reference
        proj = ds_r.GetProjection()
        proj_gt = ds_r.GetGeoTransform()

        # overwrite pixel values with 0/1 regarding their threshold value
        r_mask_data = (ds_r_val >= 0.5).astype(int)

        LOGGER.debug('Projection:' + str(proj) + '/' + 'GeoTransform:' + str(proj_gt))
        LOGGER.debug('Pixel value corner/center:' + str(r_mask_data[0, 0]) + '/' + str(r_mask_data[256, 256]))

        # set raster format definition
        raster_mask = gdal.GetDriverByName('GTiff').Create(
            raster_mask_path,  # file path
            len(r_mask_data[0]),  # width in pixels
            len(r_mask_data),  # height in pixels
            1,  # number of bands
            gdal.GDT_Float32  # type of raster
        )

        # set transformation from pixel to projected coordinates
        raster_mask.SetGeoTransform(proj_gt)

        # set projection for transformed coordinates
        raster_mask.SetProjection(proj)

        # set nodata value
        raster_mask.GetRasterBand(1).DeleteNoDataValue()
        raster_mask.GetRasterBand(1).SetNoDataValue(0)

        # write data to band 1
        raster_mask.GetRasterBand(1).WriteArray(r_mask_data)

        # flush all write cached data to disk
        raster_mask.FlushCache()

        # free and reassign
        raster_mask = None
        r_mask_data = None

        # POLYGONIZE PART ##################################################

        # file path for polygonize result
        evac_polygons_path = os.path.join(os.path.dirname(in_effects), 'evac_polygons_' + dmg_lvl.lower() + '_.gml')

        # import raster
        ds_r_mask = gdal.Open(raster_mask_path)
        ds_r_mask_band = ds_r_mask.GetRasterBand(1)

        # spatial reference
```

```
        proj = ds_r_mask.GetProjection()
        sref = osr.SpatialReference(wkt=proj)

        # set vector format definition
        src_poly = ogr.GetDriverByName("GML").CreateDataSource(evac_polygons_path)
        src_poly_lyr = src_poly.CreateLayer("evac_zone", srs=sref)

        # create polygons at pixel value 1, nodata at pixel value 0
        gdal.Polygonize(ds_r_mask_band, ds_r_mask_band, src_poly_lyr, -1, [], callback=None)

        # free and reassign
        src_poly = None
        src_poly_lyr = None

        # EVACUATION ZONE PART ##################################################

        # correction buffer because of pixel error, based on used APOLLO precision
        corr_buff = float(math.sqrt(precision ** 2 + precision ** 2))

        LOGGER.debug('Correction Buffer:' + str(corr_buff))

        # file path for evacuation zone
        evac_zone_path = os.path.join(os.path.dirname(in_effects), 'evac_zone_' + dmg_lvl.lower() + '_.gml')

        # import polygons
        src_poly = ogr.GetDriverByName("GML").Open(evac_polygons_path)
        src_poly_lyr = src_poly.GetLayer()

        # spatial reference
        sref = osr.SpatialReference()
        sref.ImportFromEPSG(epsg)

        # collect all polygons
        geom_collect = ogr.Geometry(ogr.wkbGeometryCollection)
        for feat in src_poly_lyr:
            geom_collect.AddGeometry(feat.GetGeometryRef())

        # create convex hull
        conv_hull = geom_collect.ConvexHull()
        conv_hull.AssignSpatialReference(sref)

        LOGGER.debug('Centroid as WKT:' + str(conv_hull.Centroid().ExportToWkt()))

        # set vector format definition
        src_zone = ogr.GetDriverByName("GML").CreateDataSource(evac_zone_path)
        src_zone_lyr = src_zone.CreateLayer("evac_zone", srs=sref)

        # add data to file
        field_corr_buff = ogr.FieldDefn("corr_buff", ogr.OFTReal)
        src_zone_lyr.CreateField(field_corr_buff)
        src_zone_lyr_def = src_zone_lyr.GetLayerDefn()
        conv_hull_feat = ogr.Feature(src_zone_lyr_def)
        conv_hull_feat.SetGeometry(conv_hull)
        conv_hull_feat.SetField("corr_buff", corr_buff)
        src_zone_lyr.CreateFeature(conv_hull_feat)

        # free and reassign
        conv_hull_feat = None
        src_poly = None
        src_poly_lyr = None
        src_zone = None
        src_zone_lyr = None

        # set output format and file name
        response.outputs['out_evac_zone'].output_format = Format(mime_type='text/xml', extension='.gml',
                                                         schema='http://schemas.opengis.net/gml/3.1.1/base/gml.xsd',
                                                         encoding='UTF-8', validate=None)
        response.outputs['out_evac_zone'].file = evac_zone_path
```

```
        response.outputs['out_raster'].output_format = Format(mime_type='image/geotiff', extension='.tif')
        response.outputs['out_raster'].file = raster_path

        return response
```

Listing A.9: APOLLO evacuation zone process

A.1.10 Support methods library

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/lib/geolib.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" The library is used for methods like database handling
    or spatial reference transformations.
"""

# libs
import logging
import tempfile
import psycopg2
import psycopg2.extras
from psycopg2 import sql
from pyproj import Proj, transform
from osgeo import ogr
from osgeo import osr

# authorship information
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")


# transform projection
def geo_transform(x1, y1, epsg1, epsg2):
    proj1 = Proj(init='epsg:'+str(epsg1))
    proj2 = Proj(init='epsg:'+str(epsg2))

    x2, y2 = transform(proj1, proj2, x1, y1)

    return x2, y2


# calculation of threshold distance for given solid and tnt mass
def damage_dist_threshold(tnt, solid):
    solid_dist = 0.

    # distance of float glass threshold at 3 kPa peak overpressure: R/M^(1/3)=52
    if solid == 0:
        solid_dist = 52.

    # distance of eardrum rupture threshold at 17 kPa peak overpressure: R/M^(1/3)=12.5
    if solid == 1:
        solid_dist = 12.5

```

```
    # taken from "Explosive Shocks in Air" by Graham and Kinney (Springer), derived by A. Klomfass, Fraunhofer EMI
    threshold = solid_dist * (tnt ** (1. / 3.))

    return threshold


# export spatial data from database intersected by a given geometry
def pg_export(subject, area, epsg):
    # unique geometry column identifier for general method use
    col_geom = 'geometry'

    # spatial reference
    sref = osr.SpatialReference()
    sref.ImportFromEPSG(epsg)

    # set vector format definition
    data_path = tempfile.mkstemp(prefix='db_' + subject + '_data_', suffix='.gml')[1]
    data_src = ogr.GetDriverByName("GML").CreateDataSource(data_path)
    data_lyr = data_src.CreateLayer(subject, srs=sref)

    # open database connection, using .pgpass for authentication
    if subject in ('address', 'parcel'):
        db_conn = psycopg2.connect("host=geodb port=5432 dbname=postnas_freiburg user=postgres")
    else:
        db_conn = psycopg2.connect("host=geodb port=5432 dbname=geo1 user=postgres")

    # check connection
    if db_conn is None:
        LOGGER.debug('PG connection refused.')

    # open cursor to perform database operations
    db_cur = db_conn.cursor(cursor_factory=psycopg2.extras.DictCursor)

    # sql query with placeholders and execute command, using templating mechanism for better security
    if subject == 'address':
        query = sql.SQL("SELECT st.strname AS street, ad.ha_nr AS house_nr, ST_AsText(wkb_geometry) AS {g} "
                        "FROM {ad} ad LEFT JOIN {st} st ON st.strshl = ad.strshl "
                        "WHERE ST_Intersects(ST_SetSRID(ST_PolygonFromText(%s), %s), wkb_geometry);")
        db_cur.execute(query.format(g=sql.Identifier(col_geom),
                                    ad=sql.Identifier('gdm_mat_v_haeuser'),
                                    st=sql.Identifier('str_shl')),
                       [area.ExportToWkt(), epsg])
    elif subject == 'building':
        query = sql.SQL("SELECT gmlid, lagename AS street, hausnr AS house_nr, gfk AS use_id, nutzung AS use, "
                        "klasse AS class, qualitaet AS quality, area, ST_AsText(the_geom) AS {g} "
                        "FROM {sch}.{tbl} WHERE ST_Intersects(ST_SetSRID(ST_PolygonFromText(%s), %s), the_geom);")
        db_cur.execute(query.format(g=sql.Identifier(col_geom),
                                    sch=sql.Identifier('alkis'),
                                    tbl=sql.Identifier('gebaeude')),
                       [area.ExportToWkt(), epsg])
    elif subject == 'parcel':
        query = sql.SQL("SELECT gml_id, gemarkungsnummer AS subdistrict, zaehler AS enum, nenner AS denum, "
                        "flstkz AS code, amtlicheflaeche AS area, ST_AsText(wkb_geometry) AS {g} "
                        "FROM (SELECT *, $$08$$ || {d} || $$-000-$$ || lpad({z}::text, 5, $$0$$) ||$$/$$ || "
                        "lpad(coalesce({n}, $$0$$)::text, 4, $$0$$) AS flstkz FROM {tbl} "
                        "WHERE ST_Intersects(ST_SetSRID(ST_PolygonFromText(%s), %s), wkb_geometry)) AS foo;")
        db_cur.execute(query.format(g=sql.Identifier(col_geom),
                                    d=sql.Identifier('gemarkungsnummer'),
                                    z=sql.Identifier('zaehler'),
                                    n=sql.Identifier('nenner'),
                                    tbl=sql.Identifier('ax_flurstueck')),
                       [area.ExportToWkt(), epsg])
    elif subject == 'local_plan':
        query = sql.SQL("SELECT nummer AS nr, plannr, planbez AS name, aktiv AS legal, bpplan_uid AS uid, "
                        "aenderung_von AS revision, in_kraft_datum AS date, ST_AsText(the_geom) AS {g} "
                        "FROM {sch}.{tbl} WHERE ST_Intersects(ST_SetSRID(ST_PolygonFromText(%s), %s), the_geom);")
        db_cur.execute(query.format(g=sql.Identifier(col_geom),
                                    sch=sql.Identifier('bplan'),
```

```
                                    tbl=sql.Identifier('geltungsbereich')),
                       [area.ExportToWkt(), epsg])
    elif subject == 'poi':
        query = sql.SQL("SELECT poityp, name, bezeichnung AS description, kategorie AS category, adresse AS address, "
                        "url, mail, telefon AS phone, ansprechpartner AS contact, ST_AsText(the_geom) AS {g} "
                        "FROM {sch}.{tbl} WHERE ST_Intersects(ST_SetSRID(ST_PolygonFromText(%s), %s), the_geom);")
        db_cur.execute(query.format(g=sql.Identifier(col_geom), sch=sql.Identifier('poi'), tbl=sql.Identifier('pois')),
                       [area.ExportToWkt(), epsg])

    # process query result data
    names = [desc[0] for desc in db_cur.description]
    names.remove(col_geom)
    for name in names:
        field = ogr.FieldDefn(name, ogr.OFTString)
        data_lyr.CreateField(field)

    rows = db_cur.fetchall()
    for row in rows:
        data_lyr_def = data_lyr.GetLayerDefn()
        feat = ogr.Feature(data_lyr_def)
        feat_geom = ogr.CreateGeometryFromWkt(row[col_geom])
        feat.SetGeometry(feat_geom)

        for name in names:
            feat.SetField(name, str(row[name]))

        data_lyr.CreateFeature(feat)

        # free and reassign
        feat = None

    # make the changes to the database persistent
    db_conn.commit()

    # close communication with the database
    db_cur.close()
    db_conn.close()

    # free and reassign
    db_conn = None

    return data_path
```

Listing A.10: Support methods library

A.1.11 XML parsing library

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/processes/lib/varlib.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

""" The library is used to parse the XML of WPS response documents and
    supports synchronous, asynchronous, single use and chained processes.
"""

# libs
import logging
import lxml.etree
import time
import requests

# authorship information
```

```python
__author__ = "Gunnar Ströer"
__copyright__ = "Copyright 2019, integration of wps in local sdi"
__version__ = "1.0"
__maintainer__ = "Gunnar Ströer"
__email__ = "gunnar.stroeer@yahoo.de"
__status__ = "Development"

# global variables
LOGGER = logging.getLogger("PYWPS")
VERSION = "1.0.0"
NAMESPACES = {
    'xlink': "http://www.w3.org/1999/xlink",
    'wps': "http://www.opengis.net/wps/{wps_version}",
    'ows': "http://www.opengis.net/ows/{ows_version}",
    'gml': "http://www.opengis.net/gml",
    'xsi': "http://www.w3.org/2001/XMLSchema-instance"
}

namespaces100 = {k: NAMESPACES[k].format(wps_version="1.0.0", ows_version="1.1") for k in NAMESPACES}
namespaces200 = {k: NAMESPACES[k].format(wps_version="2.0", ows_version="2.0") for k in NAMESPACES}


# return xpath namespace for given element and xpath
def get_xpath_ns(version):
    def xpath_ns(ele, path):
        if version == "1.0.0":
            nsp = namespaces100
        elif version == "2.0.0":
            nsp = namespaces200
        return ele.xpath(path, namespaces=nsp)

    return xpath_ns


# get xpath namespace
xpath_ns = get_xpath_ns(VERSION)


# return progress / result of the status response
def get_output(doc):
    process_succeeded = xpath_ns(doc, '/wps:ExecuteResponse/wps:Status/wps:ProcessSucceeded')
    process_accepted = xpath_ns(doc, '/wps:ExecuteResponse/wps:Status/wps:ProcessAccepted')
    process_status_url = xpath_ns(doc, '/wps:ExecuteResponse')
    process_status_url = process_status_url[0].attrib['statusLocation']

    LOGGER.debug('Status Reference Process Succeeded:' + str(process_succeeded))
    LOGGER.debug('Status Reference Process Accepted:' + str(process_accepted))
    LOGGER.debug('Status Reference statusLocation:' + str(process_status_url))

    # loop until statusLocation is final process result
    while not process_succeeded:
        # wait interval in seconds
        time.sleep(5)

        # reload doc from process_status_url
        r = requests.get(process_status_url, verify=False)
        doc_new = r.content

        # look for ProcessSucceeded status element
        doc = lxml.etree.fromstring(doc_new)
        process_succeeded = xpath_ns(doc, '/wps:ExecuteResponse/wps:Status/wps:ProcessSucceeded')

    result = get_output_data(doc)

    LOGGER.debug('Status Reference Result:' + str(result))

    return result
```

```python

# return the content of LiteralData, Reference or ComplexData
def get_output_data(doc):
    output = {}
    for output_el in xpath_ns(doc, '/wps:ExecuteResponse'
                                   '/wps:ProcessOutputs/wps:Output'):
        [identifier_el] = xpath_ns(output_el, './ows:Identifier')

        lit_el = xpath_ns(output_el, './wps:Data/wps:LiteralData')
        if lit_el != []:
            output[identifier_el.text] = lit_el[0].text

        ref_el = xpath_ns(output_el, './wps:Reference')
        if ref_el != []:
            LOGGER.debug('Reference XPATH:' + str(ref_el[0].attrib))
            output[identifier_el.text] = ref_el[0].attrib['{' + NAMESPACES['xlink'] + '}href']

        data_el = xpath_ns(output_el, './wps:Data/wps:ComplexData')
        if data_el != []:
            if data_el[0].text:
                output[identifier_el.text] = data_el[0].text
            else:  # XML children
                ch = list(data_el[0])[0]
                output[identifier_el.text] = lxml.etree.tostring(ch)

        # looking for BoundingBoxData
        bbox_el = xpath_ns(output_el, './ows:BoundingBox')
        if bbox_el != []:
            LOGGER.debug('BBox XPATH:' + lxml.etree.tostring(bbox_el[0]))

            output[identifier_el.text] = lxml.etree.tostring(bbox_el[0])

    return output
```

Listing A.11: XML parsing library

A.2 XML requests and responses

A.2.1 Vector intersection process request

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_vect_intersect.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute service="WPS" version="1.0.0" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd" response="document" mode="sync">
  <ows:Identifier>vect_intersect</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>in_geom_a</ows:Identifier>
      <ows:Title>Input Geometry A [gml]</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/data/evac_zone.gml" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd" method="GET" />
    </wps:Input>
    <wps:Input>
      <ows:Identifier>in_geom_b</ows:Identifier>
      <ows:Title>Input Geometry B [gml]</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/data/location_etrs.gml" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd" method="GET" />
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
```

```
    <wps:ResponseDocument lineage="false" storeExecuteResponse="false" status="false">
      <wps:Output asReference="true" mimeType="application/gml-3.1.1" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_intersect</ows:Identifier>
        <ows:Title>Intersected Geometry</ows:Title>
      </wps:Output>
    </wps:ResponseDocument>
  </wps:ResponseForm>
</wps:Execute>
```

Listing A.12: Vector intersection process request

A.2.2 Vector intersection process response

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/proc_sync_vect_intersect_response.xml

```
<!-- PyWPS 4.0.0 -->
<wps:ExecuteResponse xmlns:gml="http://www.opengis.net/gml" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US" serviceInstance="https://geodev2/pywps?service=WPS&amp;request=GetCapabilities" statusLocation="https://geodev2/wps/output/9573c328-5241-11e9-b17d-005056820f34.xml">
  <wps:Process wps:processVersion="1.0">
    <ows:Identifier>vect_intersect</ows:Identifier>
    <ows:Title>Vector Intersection Process</ows:Title>
    <ows:Abstract>The process returns intersected area of each input feature.</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2019-03-29T17:42:04Z">
    <wps:ProcessSucceeded>PyWPS Process Vector Intersection Process finished</wps:ProcessSucceeded>
  </wps:Status>
  <wps:ProcessOutputs>
    <wps:Output>
      <ows:Identifier>out_intersect</ows:Identifier>
      <ows:Title>Intersected Geometry</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/evac_zone86PgWz.gml" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd" />
    </wps:Output>
  </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

Listing A.13: Vector intersection process response

A.2.3 Quick preselection process chain request

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/chain_async_preselection.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute service="WPS" version="1.0.0" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:ogr="http://ogr.maptools.org/" xmlns:gml="http://www.opengis.net/gml" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd" response="document" mode="async">
<!-- execute proc:export_vect_data -->
  <ows:Identifier>export_vect_data</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>in_geom</ows:Identifier>
      <ows:Title>Selection Geometry [gml]</ows:Title>
      <wps:Reference mimeType="text/xml" xlink:href="https://geodev2/pywps" method="POST">
        <wps:Body>
```

```xml
        <wps:Execute service="WPS" version="1.0.0">
        <!-- execute proc:vect_buffer -->
          <ows:Identifier>vect_buffer</ows:Identifier>
          <wps:DataInputs>
            <wps:Input>
              <ows:Identifier>in_geom</ows:Identifier>
              <ows:Title>Input Geometry [gml]</ows:Title>
              <wps:Data>
                <wps:ComplexData>
                  <ogr:FeatureCollection>
                    <gml:boundedBy>
                      <gml:Box>
                        <gml:coord><gml:X>413478.281822533</gml:X><gml:Y>5316862.884733614</gml:Y></gml:coord>
                        <gml:coord><gml:X>413478.281822533</gml:X><gml:Y>5316862.884733614</gml:Y></gml:coord>
                      </gml:Box>
                    </gml:boundedBy>
                    <gml:featureMember>
                      <ogr:location fid="location.0">
                        <ogr:geometryProperty>
                          <gml:Point srsName="EPSG:25832">
                            <gml:coordinates>413478.281822533,5316862.88473361</gml:coordinates>
                          </gml:Point>
                        </ogr:geometryProperty>
                      </ogr:location>
                    </gml:featureMember>
                  </ogr:FeatureCollection>
                </wps:ComplexData>
              </wps:Data>
            </wps:Input>
            <wps:Input>
              <ows:Identifier>in_size_ref</ows:Identifier>
              <ows:Title>Buffer Size Reference</ows:Title>
              <ows:Abstract>Buffer size calculated by previous process only chainable as reference.</ows:Abstract>
              <wps:Reference mimeType="text/plain" xlink:href="https://geodev2/pywps" method="POST">
                <wps:Body>
                  <wps:Execute service="WPS" version="1.0.0">
                  <!-- execute proc:apollo_rough_dist -->
                    <ows:Identifier>apollo_rough_dist</ows:Identifier>
                    <wps:DataInputs>
                      <wps:Input>
                        <ows:Identifier>in_tnt</ows:Identifier>
                        <ows:Title>Rough TNT Blast Power [kg]</ows:Title>
                        <wps:Data>
                          <wps:LiteralData>400</wps:LiteralData>
                        </wps:Data>
                      </wps:Input>
                      <wps:Input>
                        <ows:Identifier>in_solid</ows:Identifier>
                        <ows:Title>Solid Type</ows:Title>
                        <ows:Abstract>Type of material the damage distance threshold will be calculated for: 0 = Float Glass, 1 = Eardrum Rupture</ows:Abstract>
                        <wps:Data>
                          <wps:LiteralData>0</wps:LiteralData>
                        </wps:Data>
                      </wps:Input>
                    </wps:DataInputs>
                    <wps:ResponseForm>
                      <wps:RawDataOutput>
                        <ows:Identifier>out_rough_dist</ows:Identifier>
                        <ows:Title>Rough Danger Distance</ows:Title>
                      </wps:RawDataOutput>
                    </wps:ResponseForm>
                  </wps:Execute>
                  <!-- finish proc:apollo_rough_dist -->
                </wps:Body>
              </wps:Reference>
            </wps:Input>
          </wps:DataInputs>
```

```
            <wps:ResponseForm>
              <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
                <wps:Output asReference="true" mimeType="application/gml-3.1.1" encoding="utf-8" extension=".gml">
                  <ows:Identifier>out_buff</ows:Identifier>
                  <ows:Title>Buffer Geometry</ows:Title>
                </wps:Output>
              </wps:ResponseDocument>
            </wps:ResponseForm>
          </wps:Execute>
          <!-- finish proc:vect_buffer -->
        </wps:Body>
      </wps:Reference>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>in_wfs1</ows:Identifier>
      <ows:Title>WFS Request 1 [gml]</ows:Title>
      <wps:Reference xlink:href="http://stadtplan.freiburg.de/wfs7/gdm_poi/poi_public?service=wfs&amp;version=2.0.0&amp;request=
getfeature&amp;typename=pois&amp;srsname=epsg:25832" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml
/3.1.1/base/gml.xsd" method="GET" />
    </wps:Input>
    <wps:Input>
      <ows:Identifier>in_wfs2</ows:Identifier>
      <ows:Title>WFS Request 2 [gml]</ows:Title>
      <wps:Reference xlink:href="http://stadtplan.freiburg.de/wfs7/gdm_poi/poi_public?service=wfs&amp;version=2.0.0&amp;request=
getfeature&amp;typename=pois&amp;srsname=epsg:25832&amp;Filter%3D%3CFilter%3E%3COr%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp
%3C%2FPropertyName%3E%3CLiteral%3Ekita%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp%3C%2
FPropertyName%3E%3CLiteral%3Epolizei%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp%3C%2
FPropertyName%3E%3CLiteral%3Efeuerwehr%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp%3C%2
FPropertyName%3E%3CLiteral%3Ekrankenhaeuser%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp
%3C%2FPropertyName%3E%3CLiteral%3Eschulen%3C%2FLiteral%3E%0A%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp
%3C%2FPropertyName%3E%3CLiteral%3Ebegegnung%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3C%2FOr%3E%3C%2FFilter%3E" mimeType="text/xml"
encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd" method="GET" />
    </wps:Input>
    <wps:Input>
      <ows:Identifier>in_db1</ows:Identifier>
      <ows:Title>Database Spatial Data Name 1</ows:Title>
      <ows:Abstract>Supported spatial data is defined by the following names: address, building, parcel, local_plan, poi</
ows:Abstract>
      <wps:Data>
        <wps:LiteralData>address</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>in_db2</ows:Identifier>
      <ows:Title>Database Spatial Data Name 2</ows:Title>
      <ows:Abstract>Supported spatial data is defined by the following names: address, building, parcel, local_plan, poi</
ows:Abstract>
      <wps:Data>
        <wps:LiteralData>building</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_wfs1</ows:Identifier>
        <ows:Title>WFS Request 1 Subset</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_wfs2</ows:Identifier>
        <ows:Title>WFS Request 2 Subset</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_db1</ows:Identifier>
        <ows:Title>Database Spatial Data 1 Subset</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_db2</ows:Identifier>
```

```
        <ows:Title>Database Spatial Data 2 Subset</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_bound</ows:Identifier>
        <ows:Title>Selection Boundary</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="image/geotiff" extension=".tif">
        <ows:Identifier>out_map</ows:Identifier>
        <ows:Title>Output Data Overview Map</ows:Title>
      </wps:Output>
    </wps:ResponseDocument>
  </wps:ResponseForm>
</wps:Execute>
<!-- finish proc:export_vect_data -->
```


Listing A.14: Quick preselection process chain request

A.2.4 Quick preselection process chain response status

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/chain_async_preselection_response.xml


```
<!-- PyWPS 4.0.0 -->
<wps:ExecuteResponse xmlns:gml="http://www.opengis.net/gml" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US" serviceInstance="https://geodev2/pywps?service=WPS&amp;request=GetCapabilities" statusLocation="https://geodev2/wps/output/b67e9b68-523e-11e9-b17d-005056820f34.xml">
  <wps:Process wps:processVersion="1.0">
    <ows:Identifier>export_vect_data</ows:Identifier>
    <ows:Title>Export Vector Data Process</ows:Title>
    <ows:Abstract>The process returns a subset of given or fixed spatial data selected by geometry.</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2019-03-29T17:21:32Z">
    <wps:ProcessAccepted>PyWPS Process export_vect_data accepted</wps:ProcessAccepted>
  </wps:Status>
</wps:ExecuteResponse>
```


Listing A.15: Quick preselection process chain response status

A.2.5 Quick preselection process chain response result

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/chain_async_preselection_response_status_finished.xml


```
<wps:ExecuteResponse xmlns:gml="http://www.opengis.net/gml" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US" serviceInstance="https://geodev2/pywps?service=WPS&amp;request=GetCapabilities" statusLocation="https://geodev2/wps/output/b67e9b68-523e-11e9-b17d-005056820f34.xml">
  <wps:Process wps:processVersion="1.0">
    <ows:Identifier>export_vect_data</ows:Identifier>
    <ows:Title>Export Vector Data Process</ows:Title>
    <ows:Abstract>The process returns a subset of given or fixed spatial data selected by geometry.</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2019-03-29T17:21:42Z">
    <wps:ProcessSucceeded>PyWPS Process Export Vector Data Process finished</wps:ProcessSucceeded>
  </wps:Status>
  <wps:ProcessOutputs>
    <wps:Output>
```

```
      <ows:Identifier>out_wfs1</ows:Identifier>
      <ows:Title>WFS Request 1 Subset</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/wfs_pois_data_SaFb4MQQ3Z_E.gml" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_wfs2</ows:Identifier>
      <ows:Title>WFS Request 2 Subset</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/wfs_pois_data_a2pEmiOJMJ_7.gml" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_db1</ows:Identifier>
      <ows:Title>Database Spatial Data 1 Subset</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/db_address_data_2wjJsqjmVROs.gml" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_db2</ows:Identifier>
      <ows:Title>Database Spatial Data 2 Subset</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/db_building_data_tqLsNwvBAzmY.gml" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_map</ows:Identifier>
      <ows:Title>Filtered Output Data Overview Map</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/ov_map_NEym7WVcrOwu.tif" mimeType="image/geotiff"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_bound</ows:Identifier>
      <ows:Title>Selection Boundary</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/input_aPoHx97XSeYG.gml" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    </wps:Output>
  </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

Listing A.16: Quick preselection process chain response result

A.2.6 Accurate evacuation zone process chain request

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/chain_async_main.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute service="WPS" version="1.0.0" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:ogr="http://ogr.maptools.org/" xmlns:gml="http://www.opengis.net/gml" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd" response="document" mode="async">
<!-- execute proc:export_vect_data -->
  <ows:Identifier>export_vect_data</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>in_geom</ows:Identifier>
      <ows:Title>Selection Geometry [gml]</ows:Title>
      <wps:Reference mimeType="text/xml" xlink:href="https://geodev2/pywps" method="POST">
        <wps:Body>
          <wps:Execute service="WPS" version="1.0.0">
          <!-- execute proc:vect_buffer -->
            <ows:Identifier>vect_buffer</ows:Identifier>
            <wps:DataInputs>
              <wps:Input>
                <ows:Identifier>in_geom</ows:Identifier>
                <ows:Title>Input Geometry [gml]</ows:Title>
```

```xml
                <wps:Reference mimeType="text/xml" xlink:href="https://geodev2/pywps" method="POST">
                  <wps:Body>
                    <wps:Execute service="WPS" version="1.0.0">
                    <!-- execute proc:apollo_evac_zone -->
                      <ows:Identifier>apollo_evac_zone</ows:Identifier>
                      <wps:DataInputs>
                        <wps:Input>
                          <ows:Identifier>in_conf</ows:Identifier>
                          <ows:Title>APOLLO Configuration Data [json]</ows:Title>
                          <wps:Data>
                            <wps:ComplexData>
                              {"crs": 25832, "domain": {"droi": 249.15383256726474, "zroi": 100, "name": "Ultimo"}, "bomb": {"type": "GP250", "detonator": "Front"}, "service": {"url": "https://www.cadfem.de/apollo/", "resultFile": "effects_bd769e02-2abb-11e9-92f1-005056820f34.zip"}, "geometry": {"crs": 4326, "depth": 2.7, "position": [7.840131140308953, 47.999206585002355]}, "hiddenObjects": [ "16h5647683er456", "1er5647683ef456", "1tr5647683er4r6"], "site": {"type": "Cavern", "radius": 1.5}, "height": -2.7, "mode": {"t": 50, "name": "Ultimo", "precision": 1.0}, "extent": [413229.1279899657, 5316613.730901043, 413727.4356551003, 5317112.038566177], "pitch": 0.0, "position": [413478.281822533, 5316862.88473361], "heading": 0.0}
                            </wps:ComplexData>
                          </wps:Data>
                        </wps:Input>
                        <wps:Input>
                          <ows:Identifier>in_effects</ows:Identifier>
                          <ows:Title>APOLLO Effects Result [zip|dat]</ows:Title>
                          <wps:Reference mimeType="application/octet-stream" xlink:href="https://geodev2/pywps" method="POST">
                            <wps:Body>
                              <wps:Execute service="WPS" version="1.0.0">
                              <!-- execute proc:apollo_execute -->
                                <ows:Identifier>apollo_execute</ows:Identifier>
                                <wps:DataInputs>
                                  <wps:Input>
                                    <ows:Identifier>in_conf</ows:Identifier>
                                    <ows:Title>APOLLO Configuration Data [json]</ows:Title>
                                    <wps:Reference mimeType="application/json" xlink:href="https://geodev2/pywps" method="POST">
                                      <wps:Body>
                                        <wps:Execute service="WPS" version="1.0.0">
                                        <!-- execute proc:apollo_conf -->
                                          <ows:Identifier>apollo_conf</ows:Identifier>
                                          <wps:DataInputs>
                                            <wps:Input>
                                              <ows:Identifier>in_geom</ows:Identifier>
                                              <ows:Title>Exact Location [gml]</ows:Title>
                                              <wps:Data>
                                                <wps:ComplexData>
                                                  <ogr:FeatureCollection xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://ogr.maptools.org/ evac_zone.xsd" xmlns:ogr="http://ogr.maptools.org/" xmlns:gml="http://www.opengis.net/gml">
                                                    <gml:boundedBy>
                                                      <gml:Box>
                                                        <gml:coord><gml:X>413478.281822533</gml:X><gml:Y>5316862.884733614</gml:Y></gml:coord>
                                                        <gml:coord><gml:X>413478.281822533</gml:X><gml:Y>5316862.884733614</gml:Y></gml:coord>
                                                      </gml:Box>
                                                    </gml:boundedBy>
                                                    <gml:featureMember>
                                                      <ogr:location fid="location.0">
                                                        <ogr:geometryProperty>
                                                          <gml:Point srsName="EPSG:25832">
                                                            <gml:coordinates>413478.281822533,5316862.88473361</gml:coordinates>
                                                          </gml:Point>
                                                        </ogr:geometryProperty>
                                                      </ogr:location>
                                                    </gml:featureMember>
                                                  </ogr:FeatureCollection>
                                                </wps:ComplexData>
                                              </wps:Data>
                                            </wps:Input>
                                            <wps:Input>
```

```xml
          <ows:Identifier>in_precision</ows:Identifier>
          <ows:Title>Precision [m]</ows:Title>
          <ows:Abstract>Precision used by APOLLO simulation. Supported values are: 0.5, 1.0, 2.5, 5.0, 10.0</ows:Abstract>
          <wps:Data>
            <wps:LiteralData>1.0</wps:LiteralData>
          </wps:Data>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>in_height</ows:Identifier>
          <ows:Title>Relative Height [m]</ows:Title>
          <wps:Data>
            <wps:LiteralData>-2.7</wps:LiteralData>
          </wps:Data>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>in_tnt</ows:Identifier>
          <ows:Title>Exact TNT Blast Power [kg]</ows:Title>
          <wps:Data>
            <wps:LiteralData>110</wps:LiteralData>
          </wps:Data>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>in_heading</ows:Identifier>
          <ows:Title>Bomb Azimuth Angle [deg]</ows:Title>
          <wps:Data>
            <wps:LiteralData>0.0</wps:LiteralData>
          </wps:Data>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>in_pitch</ows:Identifier>
          <ows:Title>Bomb Tilt Angle [deg]</ows:Title>
          <wps:Data>
            <wps:LiteralData>0.0</wps:LiteralData>
          </wps:Data>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>in_type</ows:Identifier>
          <ows:Title>Bomb Type</ows:Title>
          <ows:Abstract>Type of the bomb after classification. Supported values are: N/A, GP100, GP250</ows:Abstract>
          <wps:Data>
            <wps:LiteralData>GP250</wps:LiteralData>
          </wps:Data>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>in_detonator</ows:Identifier>
          <ows:Title>Detonator Position</ows:Title>
          <ows:Abstract>Position of detonator after classification. Supported values are: N/A, Front, Rear, Top, Bottom</ows:Abstract>
          <wps:Data>
            <wps:LiteralData>Front</wps:LiteralData>
          </wps:Data>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>in_site_desc</ows:Identifier>
          <ows:Title>Site Description</ows:Title>
          <ows:Abstract>Description of the bomb find location. Supported values are: Surface, Cavern</ows:Abstract>
          <wps:Data>
            <wps:LiteralData>Cavern</wps:LiteralData>
          </wps:Data>
        </wps:Input>
        <wps:Input>
          <ows:Identifier>in_site_rad</ows:Identifier>
          <ows:Title>Site Radius [m]</ows:Title>
          <wps:Data>
            <wps:LiteralData>1.5</wps:LiteralData>
```

```xml
                                              </wps:Data>
                                            </wps:Input>
                                            <wps:Input>
                                              <ows:Identifier>in_hidden</ows:Identifier>
                                              <ows:Title>Hidden Objects [gml:id1 gml:id2]</ows:Title>
                                              <ows:Abstract>List of 3D city model objects that will be ignored by the simulation. Supported values are GML identification strings.</ows:Abstract>
                                              <wps:Data>
                                                <wps:LiteralData></wps:LiteralData>
                                              </wps:Data>
                                            </wps:Input>
                                          </wps:DataInputs>
                                          <wps:ResponseForm>
                                            <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
                                              <wps:Output asReference="true" mimeType="application/json" encoding="utf-8" extension=".json">
                                                <ows:Identifier>out_conf</ows:Identifier>
                                                <ows:Title>APOLLO Configuration Data</ows:Title>
                                              </wps:Output>
                                            </wps:ResponseDocument>
                                          </wps:ResponseForm>
                                        </wps:Execute>
                                        <!-- finish proc:apollo_conf -->
                                      </wps:Body>
                                    </wps:Reference>
                                  </wps:Input>
                                  <wps:Input>
                                    <ows:Identifier>in_dem</ows:Identifier>
                                    <ows:Title>Digital Elevation Model [tif]</ows:Title>
                                    <wps:Reference mimeType="image/geotiff" xlink:href="https://geodev2/pywps" method="POST">
                                      <wps:Body>
                                        <wps:Execute service="WPS" version="1.0.0">
                                        <!-- execute proc:export_3d_data -->
                                          <ows:Identifier>export_3d_data</ows:Identifier>
                                          <wps:DataInputs>
                                            <wps:Input>
                                              <ows:Identifier>in_geom</ows:Identifier>
                                              <ows:Title>Selection Geometry [gml]</ows:Title>
                                              <wps:Reference mimeType="text/xml" xlink:href="https://geodev2/pywps" method="POST">
                                                <wps:Body>
                                                  <wps:Execute service="WPS" version="1.0.0">
                                                  <!-- execute proc:vect_buffer -->
                                                    <ows:Identifier>vect_buffer</ows:Identifier>
                                                    <wps:DataInputs>
                                                      <wps:Input>
                                                        <ows:Identifier>in_geom</ows:Identifier>
                                                        <ows:Title>Input Geometry [gml]</ows:Title>
                                                        <wps:Data>
                                                          <wps:ComplexData>
                                                            <ogr:FeatureCollection>
                                                              <gml:boundedBy>
                                                                <gml:Box>
                                                                  <gml:coord><gml:X>413478.281822533</gml:X><gml:Y>5316862.884733614</gml:Y></gml:coord>
                                                                  <gml:coord><gml:X>413478.281822533</gml:X><gml:Y>5316862.884733614</gml:Y></gml:coord>
                                                                </gml:Box>
                                                              </gml:boundedBy>
                                                              <gml:featureMember>
                                                                <ogr:location fid="location.0">
                                                                  <ogr:geometryProperty>
                                                                    <gml:Point srsName="EPSG:25832">
                                                                      <gml:coordinates>413478.281822533,5316862.88473361</gml:coordinates>
                                                                    </gml:Point>
                                                                  </ogr:geometryProperty>
                                                                </ogr:location>
                                                              </gml:featureMember>
```

```xml
                                                            </ogr:FeatureCollection>
                                                          </wps:ComplexData>
                                                        </wps:Data>
                                                      </wps:Input>
                                                      <wps:Input>
                                                        <ows:Identifier>in_size_ref</ows:Identifier>
                                                        <ows:Title>Buffer Size Reference</ows:Title>
                                                        <ows:Abstract>Buffer size calculated by previous process only chainable as reference.</ows:Abstract>
                                                        <wps:Reference mimeType="text/plain" xlink:href="https://geodev2/pywps" method="POST">
                                                          <wps:Body>
                                                            <wps:Execute service="WPS" version="1.0.0">
                                                            <!-- execute proc:apollo_rough_dist -->
                                                              <ows:Identifier>apollo_rough_dist</ows:Identifier>
                                                              <wps:DataInputs>
                                                                <wps:Input>
                                                                  <ows:Identifier>in_tnt</ows:Identifier>
                                                                  <ows:Title>Rough TNT Blast Power [kg]</ows:Title>
                                                                  <wps:Data>
                                                                    <wps:LiteralData>110</wps:LiteralData>
                                                                  </wps:Data>
                                                                </wps:Input>
                                                                <wps:Input>
                                                                  <ows:Identifier>in_solid</ows:Identifier>
                                                                  <ows:Title>Solid Type</ows:Title>
                                                                  <ows:Abstract>Type of material the damage distance threshold will be calculated for: 0 = Float Glass, 1 = Eardrum Rupture</ows:Abstract>
                                                                  <wps:Data>
                                                                    <wps:LiteralData>0</wps:LiteralData>
                                                                  </wps:Data>
                                                                </wps:Input>
                                                              </wps:DataInputs>
                                                              <wps:ResponseForm>
                                                                <wps:RawDataOutput>
                                                                  <ows:Identifier>out_rough_dist</ows:Identifier>
                                                                  <ows:Title>Rough Danger Distance</ows:Title>
                                                                </wps:RawDataOutput>
                                                              </wps:ResponseForm>
                                                            </wps:Execute>
                                                            <!-- finish proc:apollo_rough_dist -->
                                                          </wps:Body>
                                                        </wps:Reference>
                                                      </wps:Input>
                                                    </wps:DataInputs>
                                                    <wps:ResponseForm>
                                                      <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
                                                        <wps:Output asReference="true" mimeType="application/gml-3.1.1" encoding="utf-8" extension=".gml">
                                                          <ows:Identifier>out_buff</ows:Identifier>
                                                          <ows:Title>Buffer Geometry</ows:Title>
                                                        </wps:Output>
                                                      </wps:ResponseDocument>
                                                    </wps:ResponseForm>
                                                  </wps:Execute>
                                                  <!-- finish proc:vect_buffer -->
                                                </wps:Body>
                                              </wps:Reference>
                                            </wps:Input>
                                          </wps:DataInputs>
                                          <wps:ResponseForm>
                                            <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
                                              <wps:Output asReference="true" mimeType="image/geotiff" extension=".tif">
                                                <ows:Identifier>out_dem</ows:Identifier>
                                                <ows:Title>Digital Elevation Model</ows:Title>
                                              </wps:Output>
                                            </wps:ResponseDocument>
```

```xml
        </wps:ResponseForm>
      </wps:Execute>
      <!-- finish proc:export_3d_data -->
    </wps:Body>
  </wps:Reference>
</wps:Input>
<wps:Input>
  <ows:Identifier>in_city</ows:Identifier>
  <ows:Title>3D City Model [x3d]</ows:Title>
  <wps:Reference mimeType="text/xml" xlink:href="https://geodev2/pywps" method="POST">
    <wps:Body>
      <wps:Execute service="WPS" version="1.0.0">
      <!-- execute proc:export_3d_data -->
        <ows:Identifier>export_3d_data</ows:Identifier>
        <wps:DataInputs>
          <wps:Input>
            <ows:Identifier>in_geom</ows:Identifier>
            <ows:Title>Selection Geometry [gml]</ows:Title>
            <wps:Reference mimeType="text/xml" xlink:href="https://geodev2/pywps" method="POST">
              <wps:Body>
                <wps:Execute service="WPS" version="1.0.0">
                <!-- execute proc:vect_buffer -->
                  <ows:Identifier>vect_buffer</ows:Identifier>
                  <wps:DataInputs>
                    <wps:Input>
                      <ows:Identifier>in_geom</ows:Identifier>
                      <ows:Title>Input Geometry [gml]</ows:Title>
                      <wps:Data>
                        <wps:ComplexData>
                          <ogr:FeatureCollection>
                            <gml:boundedBy>
                              <gml:Box>
                                <gml:coord><gml:X>413478.281822533</gml:X><gml:Y>5316862.884733614</gml:Y></gml:coord>
                                <gml:coord><gml:X>413478.281822533</gml:X><gml:Y>5316862.884733614</gml:Y></gml:coord>
                              </gml:Box>
                            </gml:boundedBy>
                            <gml:featureMember>
                              <ogr:location fid="location.0">
                                <ogr:geometryProperty>
                                  <gml:Point srsName="EPSG:25832">
                                    <gml:coordinates>413478.281822533,5316862.88473361</gml:coordinates>
                                  </gml:Point>
                                </ogr:geometryProperty>
                              </ogr:location>
                            </gml:featureMember>
                          </ogr:FeatureCollection>
                        </wps:ComplexData>
                      </wps:Data>
                    </wps:Input>
                    <wps:Input>
                      <ows:Identifier>in_size_ref</ows:Identifier>
                      <ows:Title>Buffer Size Reference</ows:Title>
                      <ows:Abstract>Buffer size calculated by previous process only chainable as reference.</ows:Abstract>
                      <wps:Reference mimeType="text/plain" xlink:href="https://geodev2/pywps" method="POST">
                        <wps:Body>
                          <wps:Execute service="WPS" version="1.0.0">
                          <!-- execute proc:apollo_rough_dist -->
                            <ows:Identifier>apollo_rough_dist</ows:Identifier>
                            <wps:DataInputs>
                              <wps:Input>
                                <ows:Identifier>in_tnt</ows:Identifier>
                                <ows:Title>Rough TNT Blast Power [kg]</ows:Title>
                                <wps:Data>
```

```
                                                                    <wps:LiteralData>110</wps:LiteralData>
                                                                  </wps:Data>
                                                                </wps:Input>
                                                                <wps:Input>
                                                                  <ows:Identifier>in_solid</ows:Identifier>
                                                                  <ows:Title>Solid Type</ows:Title>
                                                                  <ows:Abstract>Type of material the damage distance threshold will be calculated for: 0 = Float Glass, 1 = Eardrum Rupture</ows:Abstract>
                                                                  <wps:Data>
                                                                    <wps:LiteralData>0</wps:LiteralData>
                                                                  </wps:Data>
                                                                </wps:Input>
                                                              </wps:DataInputs>
                                                              <wps:ResponseForm>
                                                                <wps:RawDataOutput>
                                                                  <ows:Identifier>out_rough_dist</ows:Identifier>
                                                                  <ows:Title>Rough Danger Distance</ows:Title>
                                                                </wps:RawDataOutput>
                                                              </wps:ResponseForm>
                                                            </wps:Execute>
                                                            <!-- finish proc:apollo_rough_dist -->
                                                          </wps:Body>
                                                        </wps:Reference>
                                                      </wps:Input>
                                                    </wps:DataInputs>
                                                    <wps:ResponseForm>
                                                      <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
                                                        <wps:Output asReference="true" mimeType="application/gml-3.1.1" encoding="utf-8" extension=".gml">
                                                          <ows:Identifier>out_buff</ows:Identifier>
                                                          <ows:Title>Buffer Geometry</ows:Title>
                                                        </wps:Output>
                                                      </wps:ResponseDocument>
                                                    </wps:ResponseForm>
                                                  </wps:Execute>
                                                  <!-- finish proc:vect_buffer -->
                                                </wps:Body>
                                              </wps:Reference>
                                            </wps:Input>
                                          </wps:DataInputs>
                                          <wps:ResponseForm>
                                            <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
                                              <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".x3d">
                                              <ows:Identifier>out_city</ows:Identifier>
                                              <ows:Title>3D City Model</ows:Title>
                                            </wps:Output>
                                            </wps:ResponseDocument>
                                          </wps:ResponseForm>
                                        </wps:Execute>
                                        <!-- finish proc:export_3d_data -->
                                      </wps:Body>
                                    </wps:Reference>
                                  </wps:Input>
                                </wps:DataInputs>
                                <wps:ResponseForm>
                                  <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
                                    <wps:Output asReference="true" mimeType="application/octet-stream">
                                      <ows:Identifier>out_effects</ows:Identifier>
                                      <ows:Title>APOLLO Effects Result</ows:Title>
                                    </wps:Output>
                                  </wps:ResponseDocument>
                                </wps:ResponseForm>
                              </wps:Execute>
                              <!-- finish proc:apollo_execute -->
                            </wps:Body>
                          </wps:Reference>
                        </wps:Input>
```

```xml
                        <wps:Input>
                          <ows:Identifier>in_dmg_lvl</ows:Identifier>
                          <ows:Title>Damage Level</ows:Title>
                          <ows:Abstract>Level of damage the evacuation zone will be calculated for: 0 = Float Glass, 1 = Hardened Glass, 2 = Safety Glass, 3 = Masonry, 4 = Eardrum Rupture, 5 = Injury, 6 = Lethal Injury</ows:Abstract>
                          <wps:Data>
                            <wps:LiteralData>0</wps:LiteralData>
                          </wps:Data>
                        </wps:Input>
                      </wps:DataInputs>
                      <wps:ResponseForm>
                        <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
                          <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
                            <ows:Identifier>out_evac_zone</ows:Identifier>
                            <ows:Title>Evacuation Zone</ows:Title>
                          </wps:Output>
                          <wps:Output asReference="true" mimeType="image/geotiff" extension=".tif">
                            <ows:Identifier>out_raster</ows:Identifier>
                            <ows:Title>Evacuation Raster</ows:Title>
                          </wps:Output>
                        </wps:ResponseDocument>
                      </wps:ResponseForm>
                    </wps:Execute>
                    <!-- finish proc:apollo_evac_zone -->
                  </wps:Body>
                </wps:Reference>
              </wps:Input>
              <wps:Input>
                <ows:Identifier>in_size_field</ows:Identifier>
                <ows:Title>Buffer Size Field Name</ows:Title>
                <ows:Abstract>Name of input geometry attribute field which value will be used for buffer size.</ows:Abstract>
                <wps:Data>
                  <wps:LiteralData>corr_buff</wps:LiteralData>
                </wps:Data>
              </wps:Input>
            </wps:DataInputs>
            <wps:ResponseForm>
              <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
                <wps:Output asReference="true" mimeType="application/gml-3.1.1" encoding="utf-8" extension=".gml">
                  <ows:Identifier>out_buff</ows:Identifier>
                  <ows:Title>Buffer Geometry</ows:Title>
                </wps:Output>
              </wps:ResponseDocument>
            </wps:ResponseForm>
          </wps:Execute>
          <!-- finish proc:vect_buffer -->
        </wps:Body>
      </wps:Reference>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>in_wfs1</ows:Identifier>
      <ows:Title>WFS Request 1 [gml]</ows:Title>
      <wps:Reference xlink:href="http://stadtplan.freiburg.de/wfs7/gdm_poi/poi_public?service=wfs&amp;version=2.0.0&amp;request=getfeature&amp;typename=pois&amp;srsname=epsg:25832" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd" method="GET" />
    </wps:Input>
    <wps:Input>
      <ows:Identifier>in_wfs2</ows:Identifier>
      <ows:Title>WFS Request 2 [gml]</ows:Title>
      <wps:Reference xlink:href="http://stadtplan.freiburg.de/wfs7/gdm_poi/poi_public?service=wfs&amp;version=2.0.0&amp;request=getfeature&amp;typename=pois&amp;srsname=epsg:25832&amp;Filter%3D%3CFilter%3E%3COr%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp%3C%2FPropertyName%3E%3CLiteral%3Ekita%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp%3C%2FPropertyName%3E%3CLiteral%3Epolizei%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp%3C%2FPropertyName%3E%3CLiteral%3Efeuerwehr%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp%3C%2FPropertyName%3E%3CLiteral%3Ekrankenhaeuser%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp%3C%2FPropertyName%3E%3CLiteral%3Eschulen%3C%2FLiteral%3E%0A%3C%2FPropertyIsEqualTo%3E%3CPropertyIsEqualTo%3E%3CPropertyName%3Epoityp%3C%2FPropertyName%3E%3CLiteral%3Ebegegnung%3C%2FLiteral%3E%3C%2FPropertyIsEqualTo%3E%3C%2FOr%3E%3C%2FFilter%3E" mimeType="text/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd" method="GET" />
```

```
    </wps:Input>
    <wps:Input>
      <ows:Identifier>in_db1</ows:Identifier>
      <ows:Title>Database Spatial Data Name 1</ows:Title>
      <ows:Abstract>Supported spatial data is defined by the following names: address, building, parcel, local_plan, poi</ows:Abstract>
      <wps:Data>
        <wps:LiteralData>address</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>in_db2</ows:Identifier>
      <ows:Title>Database Spatial Data Name 2</ows:Title>
      <ows:Abstract>Supported spatial data is defined by the following names: address, building, parcel, local_plan, poi</ows:Abstract>
      <wps:Data>
        <wps:LiteralData>building</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:ResponseDocument lineage="false" storeExecuteResponse="true" status="true">
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_wfs1</ows:Identifier>
        <ows:Title>WFS Request 1 Subset</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_wfs2</ows:Identifier>
        <ows:Title>WFS Request 2 Subset</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_db1</ows:Identifier>
        <ows:Title>Database Spatial Data 1 Subset</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_db2</ows:Identifier>
        <ows:Title>Database Spatial Data 2 Subset</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="text/xml" encoding="utf-8" extension=".gml">
        <ows:Identifier>out_bound</ows:Identifier>
        <ows:Title>Selection Boundary</ows:Title>
      </wps:Output>
      <wps:Output asReference="true" mimeType="image/geotiff" extension=".tif">
        <ows:Identifier>out_map</ows:Identifier>
        <ows:Title>Output Data Overview Map</ows:Title>
      </wps:Output>
    </wps:ResponseDocument>
  </wps:ResponseForm>
</wps:Execute>
<!-- finish proc:export_vect_data -->
```

Listing A.17: Accurate evacuation zone process chain request

A.2.7 Accurate evacuation zone process chain response status

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/chain_async_main_response.xml

```
<!-- PyWPS 4.0.0 -->
<wps:ExecuteResponse xmlns:gml="http://www.opengis.net/gml" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US" serviceInstance="https://geodev2/pywps?service=WPS&amp;request=GetCapabilities" statusLocation="https://geodev2/wps/output/06670340-523f-11e9-8bcd-005056820f34.xml">
```

```xml
  <wps:Process wps:processVersion="1.0">
    <ows:Identifier>export_vect_data</ows:Identifier>
    <ows:Title>Export Vector Data Process</ows:Title>
    <ows:Abstract>The process returns a subset of given or fixed spatial data selected by geometry.</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2019-03-29T17:23:47Z">
    <wps:ProcessAccepted>PyWPS Process export_vect_data accepted</wps:ProcessAccepted>
  </wps:Status>
</wps:ExecuteResponse>
```


Listing A.18: Accurate evacuation zone process chain response status

A.2.8 Accurate evacuation zone process chain response result

https://gitlab.com/hadlaskard/integration-of-wps-in-local-sdi/blob/master/xml/chain_async_main_response_status_finished.xml


```xml
<wps:ExecuteResponse xmlns:gml="http://www.opengis.net/gml" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.opengis.
net/wps/1.0.0" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang
="en-US" serviceInstance="https://geodev2/pywps?service=WPS&amp;request=GetCapabilities" statusLocation="https://geodev2/wps/output
/06670340-523f-11e9-8bcd-005056820f34.xml">
  <wps:Process wps:processVersion="1.0">
    <ows:Identifier>export_vect_data</ows:Identifier>
    <ows:Title>Export Vector Data Process</ows:Title>
    <ows:Abstract>The process returns a subset of given or fixed spatial data selected by geometry.</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2019-03-29T17:25:14Z">
    <wps:ProcessSucceeded>PyWPS Process Export Vector Data Process finished</wps:ProcessSucceeded>
  </wps:Status>
  <wps:ProcessOutputs>
    <wps:Output>
      <ows:Identifier>out_wfs1</ows:Identifier>
      <ows:Title>WFS Request 1 Subset</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/wfs_pois_data_vziWmnppvthQ.gml" mimeType="text/xml" encoding="UTF-8"
schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_wfs2</ows:Identifier>
      <ows:Title>WFS Request 2 Subset</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/wfs_pois_data_ZKNowN2_LMTj.gml" mimeType="text/xml" encoding="UTF-8"
schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_db1</ows:Identifier>
      <ows:Title>Database Spatial Data 1 Subset</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/db_address_data_9XpzUAsIAWWF.gml" mimeType="text/xml" encoding="UTF-8"
schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_db2</ows:Identifier>
      <ows:Title>Database Spatial Data 2 Subset</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/db_building_data_TODOKrEDToO1.gml" mimeType="text/xml" encoding="UTF-8"
schema="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_map</ows:Identifier>
      <ows:Title>Filtered Output Data Overview Map</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/ov_map_eC7jioMPBgxx.tif" mimeType="image/geotiff"/>
    </wps:Output>
    <wps:Output>
      <ows:Identifier>out_bound</ows:Identifier>
      <ows:Title>Selection Boundary</ows:Title>
      <wps:Reference xlink:href="https://geodev2/wps/output/input_BE18QyJZZi5v.gml" mimeType="text/xml" encoding="UTF-8" schema="http:
//schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
```

```
    </wps:Output>
  </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

Listing A.19: Accurate evacuation zone process chain response result