International Journal of Geo-Information

MDPI

Article

Formalizing Parameter Constraints to Support Intelligent Geoprocessing: A SHACL-Based Method

Zhi-Wei Hou [1,2], Cheng-Zhi Qin [1,2,3,*], A-Xing Zhu [1,2,3,4,5], Yi-Jie Wang [1,2], Peng Liang [6], Yu-Jing Wang [1,2] and Yun-Qiang Zhu [1,2,3]

1 State Key Laboratory of Resources and Environmental Information System, Institute of Geographic Sciences and Natural Resources Research, CAS, Beijing 100101, China; houzw@lreis.ac.cn (Z.-W.H.); azhu@wisc.edu (A.-X.Z.); wangyijie@lreis.ac.cn (Y.-J.W.); wangyujing@lreis.ac.cn (Y.-J.W.); zhuyq@lreis.ac.cn (Y.-Q.Z.)
2 College of Resources and Environment, University of Chinese Academy of Sciences, Beijing 100049, China
3 Jiangsu Center for Collaborative Innovation in Geographical Information Resource Development and Application, Nanjing 210023, China
4 School of Geography, Nanjing Normal University, Nanjing 210023, China
5 Department of Geography, University of Wisconsin-Madison, Madison, WI 53706, USA
6 Institute of Earthquake Forecasting, The China Earthquake Administration, Beijing 100036, China; liangp@lreis.ac.cn
* Correspondence: qincz@lreis.ac.cn; Tel.: +86-010-6488-8959

Abstract: Intelligent geoprocessing relies heavily on formalized parameter constraints of geoprocessing tools to validate the input data and to further ensure the robustness and reliability of geoprocessing. However, existing methods developed to formalize parameter constraints are either designed based on ill-suited assumptions, which may not correctly identify the invalid parameter inputs situation, or are inefficient to use. This paper proposes a novel method to formalize the parameter constraints of geoprocessing tools, based on a high-level and standard constraint language (i.e., SHACL) and geoprocessing ontologies, under the guidance of a systematic classification of parameter constraints. An application case and a heuristic evaluation were conducted to demonstrate and evaluate the effectiveness and usability of the proposed method. The results show that the proposed method is not only comparatively easier and more efficient than existing methods but also covers more types of parameter constraints, for example, the application-context-matching constraints that have been ignored by existing methods.

Keywords: intelligent geoprocessing; parameter constraints; input data validation; formalization; heuristic evaluation; application context

Citation: Hou, Z.-W.; Qin, C.-Z.; Zhu, A.-X.; Wang, Y.-J.; Liang, P.; Wang, Y.-J.; Zhu, Y.-Q. Formalizing Parameter Constraints to Support Intelligent Geoprocessing: A SHACL-Based Method. *ISPRS Int. J. Geo-Inf.* **2021**, *10*, 605. https://doi.org/10.3390/ijgi10090605

Academic Editors: Rob Brennan, Brian Davis, Armin Haller, Beyza Yaman and Wolfgang Kainz

Received: 2 July 2021
Accepted: 6 September 2021
Published: 14 September 2021

1. Introduction

In the last two decades, intelligent geoprocessing based on Semantic Web technologies such as the Resource Description Framework (RDF) and ontology has shown promise to significantly improve the efficiency and effectiveness of the discovery, composition, and execution of geoprocessing tools (especially geoprocessing web services) [1–6]. Among the research topics in intelligent geoprocessing, parameter constraints of geoprocessing tools have been increasingly recognized as a key factor in the success of intelligent geoprocessing [2,3,7–10].

The parameter constraints of a geoprocessing tool are a group of pre/post-conditions that must be satisfied by the input data of the geoprocessing tool's parameters (Figure 1a,b), such as the cardinality constraints which are pre-conditions on the occurrence number of inputs. Here the parameters could be inputs, outputs, or options of a geoprocessing tool. Parameter constraints are essential in the validation and further refinement of the input data of the parameters (called parameter inputs hereafter) which might be incorrect, incomplete, inconsistent, or inappropriate for geoprocessing [3,9–11]. Moreover, parameter

constraints are important in the discovery and composition of the correct tools to satisfy the goals of end-users or to improve the robustness of geoprocessing workflows [2,3,7,8].

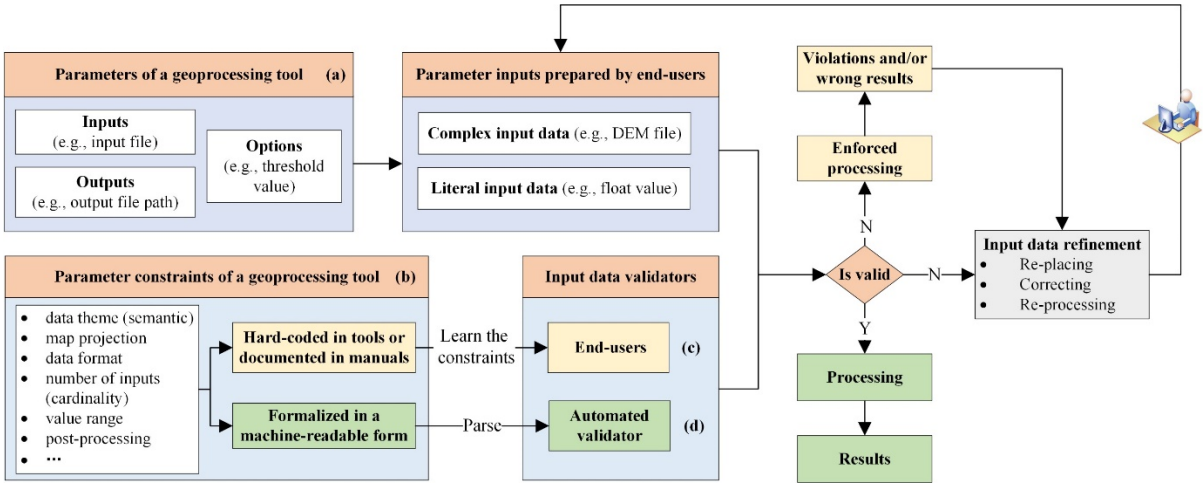

Figure 1. The roles of parameter constraints and input data validation in geoprocessing. (**a**) Different types of parameters of a geoprocessing tool; (**b**) examples of parameter constraints and how to use them for validations; (**c**) input data validation depends on users; (**d**) automated input data validation using specifically designed validators.

Currently, most of the parameter constraints are hard-coded in geoprocessing tools and/or documented in their manuals (including extended process profiles of web services [10]). As a result, intelligent geoprocessing relies heavily on end-users' understanding of the tools and trial-and-errors to validate the parameter inputs [2,7,8,12] (Figure 1c). This is frustrating and time-consuming for end-users, especially for non-experts. Additionally, it often cannot properly detect the semantic violations raised from parameter inputs.

Thus, automated semantic validation of parameter inputs (Figure 1d) is needed urgently in intelligent geoprocessing, especially as the number of available yet heterogeneous geoprocessing tools and datasets are dramatically and continually growing. The fundamental research issue for automated validation of parameter inputs is how to explicitly formalize the parameter constraints into a machine-readable form (typically, RDF graphs), so to support intelligent geoprocessing.

1.1. Existing Methods of Formalizing Parameter Constraints

Existing methods to formalize parameter constraints can be classified into two categories according to the adopted representation language and how they validate input data, namely, inference-oriented methods and query-oriented methods. The inference-oriented methods represent parameter constraints as ontology restrictions (e.g., owl:minCardinality) by using ontology languages (such as the Web Ontology Language, OWL) [13–15] and/or inference rules [3,7,8,16] by using rule languages (like the Semantic Web Rule Language, SWRL). Then, the conformities of parameter inputs that have been explicitly encoded as RDF graphs can be validated against these restrictions or rules through logical inferences. If the inferences fail or produce conflicts, the input data will be identified as invalid. Currently, inference-oriented methods are frequently used in intelligent geoprocessing since the languages they adopt are widely used in the representation of the semantics of geoprocessing tools and data.

However, input data validation based on the parameter constraints formalized by the inference-oriented methods is unable to detect all the kinds of violations, in particular incompleteness. This is because the languages used in these methods are designed under the Open-World Assumption (OWA) and Non-Unique Name Assumption (NUNA) [17–19]. OWA assumes that the information within a system is incomplete, thus, what has not been explicitly asserted to be true is unknown for true or false; NUNA assumes that the same

entity may have different names [18]. Whereas, a qualified validation of input data requires the languages for constraint formalization to be designed under the Closed-World Assumption (CWA) and Unique Name Assumption (UNA). CWA assumes that the information within a system is complete so that what has not been asserted to be true must be false; UNA assumes that an entity can only have one unique name [18]. Table 1 exemplifies the differences between the validation results under these two types of assumptions. The cardinality constraint used in Table 1 restricts that the parameter must have and only have one input. However, parameter inputs are always valid under OWA and NUNA when using inference-oriented methods, no matter how many inputs are provided. Only the validations under CWA and UNA can correctly detect the incompleteness of inputs. This means the inference-oriented methods are not suitable for the formalization of parameter constraints in intelligent geoprocessing. Note that the inference-oriented methods discussed in this study are different from those methods which focus on extending standard OWL, RDF/RDFS, or description logics to express and validate constraint semantics [17,20,21]. The latter methods are designed based on closed-world settings, thus may be suitable for formalizing parameter constraints. However, to the best of our knowledge, they have not been used so far to formalize the parameter constraints of geoprocessing tools, which is the focus of this study.

Table 1. The differences in the results of data validation under OWA-NUNA versus those under CWA-UNA against a cardinality constraint that the parameter must have only one input.

Number of Inputs for Validation	Under OWA and NUNA		Under CWA and UNA	
	Is Valid	Reason	Is Valid	Reason
no input (incomplete input)	true	infer that there might have one or more unknown inputs	false	missing required input
only one input (as the required)	true	it exactly has one required input	true	it exactly has one required input
two different inputs (too many inputs)	true	infer that the two inputs are the same entity with different names *	false	can only have one input, but provided two different inputs

* Note: Here we suppose these two inputs have not been explicitly declared to be different using OWL restriction properties such as the owl: differentFrom.

The second category of methods to formalize parameter constraints is the query-oriented methods [22,23], which represent parameter constraints as query statements using RDF query languages such as SPARQL (short for SPARQL Protocol And RDF Query Language) [24] and its subset $\text{SPARQL}_{\text{CONSTRAINT}}$ [23]. Here, violations within the input data can be filtered out by comparing the properties of input data with the query conditions. Since SPARQL is an expressive language designed under CWA and UNA, validation based on the parameter constraints formalized by the query-oriented methods, or so-called SPARQL-based methods, can correctly detect all the possible violations, including incompleteness.

1.2. Research Question in This Study

However, these existing query-oriented methods are difficult and inefficient (i.e., low in usability) to learn, use, and maintain by users, including knowledge engineers who are familiar with SPARQL (or its subsets), but not with geoprocessing tools, and the developers of geoprocessing tools in the opposite situation. The first reason is that these methods lack explicit guidelines that users can use to easily identify and formalize all the possible types of parameter constraints a tool might have. As a result, users (especially knowledge engineers who are unfamiliar with geoprocessing tools) might ignore some of the parameter constraints, which could eventually lead to violations. This is particularly true in the case of application domain-specific constraints that mainly exist in the minds of experts. The second reason is that SPARQL and its subset are low-level languages that are primarily designed to query RDF data. This implies that they lack standardized

high-level abstraction of constraint concepts that are concise and intuitive for users to understand and use [19,25]. Although there are some tools (e.g., RDFUnit [26]) that could facilitate the definition of SPARQL-based constraints, users (even those knowledge engineers) still require a considerable amount of time and effort to translate the in-mind parameter constraints into verbose and complex SPARQL queries, as well as to explicitly write the implementation details to compare the constraints with input data properties. For example, to formalize the cardinality constraints of a single parameter, one needs to write several SPARQL queries to accomplish the following tasks: (1) select the target parameter and its input datasets; (2) count the number of input datasets; (3) compare the counted number with the condition (i.e., the required number); and (4) define end-user-friendly messages to report the validation results, to help end-users remove the violations.

Therefore, how to easily and efficiently formalize all the possible types of parameter constraints for a geoprocessing tool is still a crucial problem in intelligent geoprocessing. To address this problem, this study proposes a method of formalizing parameter constraints based on a standard high-level RDF constraint language, i.e., the Shapes Constraint Language (SHACL) [27]. A comprehensive analysis and classification of parameter constraints (Section 2) were conducted to guide the design and implementation of the method. Then, Section 3 presents the basic idea of the proposed method and a detailed design in two key steps: (1) identifying different types of parameter constraints following a two-phase process, and (2) formalizing each type of the obtained parameter constraints based on SHACL and ontologies. Section 4 offers an application case to illustrate how the proposed method could be used to formalize the parameter constraints of a geoprocessing tool. Section 5 provides a usability evaluation of the proposed method, which is compared with existing methods. Results of the application case and usability evaluation show that the proposed method, compared with existing method, is comparatively easier to use, and more efficient when covering more types of parameter constraints. Section 6 summarizes the main conclusions and possible future developments.

2. Parameter Constraints of Geoprocessing Tools

A geoprocessing tool may contain a variety of syntactic and semantic parameter constraints, ranging from commonly used constraints (such as cardinality and value range) to complex and less common constraints that relate to multiple parameters or the application context. These constraints may exist in different forms, including structured documentation and domain expertise. For example, the suitable distance value ranges for the buffering analysis in different application contexts usually lack explicit documentation. As a result, it is very difficult for users who are responsible for formalization to completely and correctly identify and formalize all the possible parameter constraints that a tool might have. Thus, to better understand the parameter constraints and to further guide the design and implementation of the formalization methods, a high-level analysis of the appropriate classification of parameter constraints is conducted in this section. It clarifies the concepts and relationships, as well as the characteristics of different types of parameter constraints, and provides principle means of formalizing and organizing the constraints coherently and cohesively.

The constraints of a specific parameter of a geoprocessing tool typically consist of three parts: (1) the validation targets, i.e., the parameter inputs that need to be validated, (2) the detailed pre/post-conditions on input data properties of the targets, and (3) violation feedback, including the violation severity and user-friendly messages that could help end-users to understand and fix the violations [28,29]. Each part can be further divided into different levels or types based on the application scopes, data properties, and severities, respectively (Figure 2).

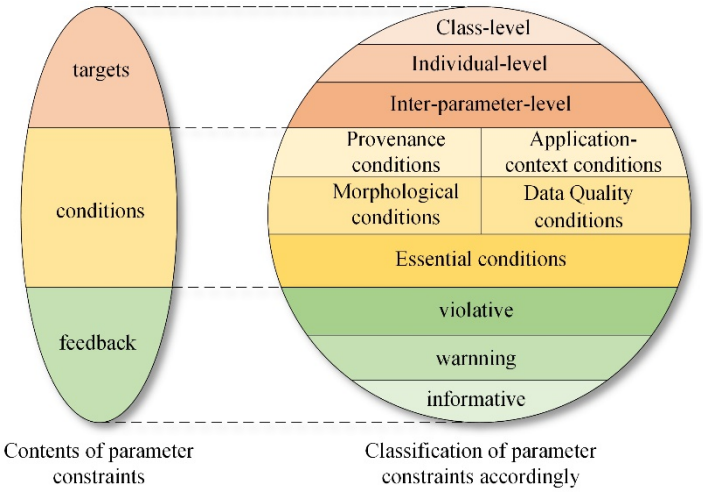

Figure 2. Classification of parameter constraints according to their contents.

Note that feedback is optional in many cases because the validators typically could generate the information automatically. For example, SimpleConsist [28] (a constraint checker) will yield the witnesses of a failure if the data do not conform to an ontology; and the ontology evaluation results generated in OOPS! [29] include not only the detected pitfalls (i.e., problematic ontology characteristics) but also their importance levels. Therefore, the following sub-sections focus on the conceptualization and classification of parameter constraints from the aspect of their targets and conditions, respectively.

2.1. Target-Oriented Classification of Parameter Constraints

Parameter constraints can be divided into three different levels according to the application scope of the validation targets (Figure 2 and Table 2).

Table 2. Classification of parameter constraints of geoprocessing tools according to their validation targets.

Constraint Types (Based on Validation Targets)	Subtypes	Description
Class-level constraints	-	the constraints on a group of targets (e.g., the instances of a target class or subjects/objects of a target predicate).
Individual-level constraints	parameter-level data-level	constraints on the input data of each parameter as a whole. constraints on each input data property.
Inter-parameter-level constraints	equivalent constraints dependency constraints	inputs of two (or more) given parameters must conform to the same conditions: e.g., the coordinate reference system. conditions on the inputs of a parameter are determined by the properties of input data of another parameter.

Class-level constraints refer to the constraints that could be applied to a group of targets which are either instances of an ontology class or are subjects/objects of a specific predicate. For example, the cardinality constraint on the data theme property of all raster datasets which are instances of class "data:RasterData" in the data ontology. Class-level constraints can be used to validate the input data automatically so long as the users define the datasets as instances or subjects/objects of the targeted class or predicate. Thus, the definition of class-level constraints could reduce repetitive formalization work.

On the contrary, individual-level constraints have only one specific validation target. An example of this is the input dataset for the parameter "slope" of the WetnessIndex tool in WhiteboxTools [30]. Individual-level constraints can be divided into two sub-types. The first is the parameter-level constraints which are primarily the value types and cardinality conditions on the input data of a specific parameter as a whole. The second is data-level

constraints which put restrictions on the input data properties (such as the data type and value range) of a particular parameter. Individual-level constraints are the most widely used type of parameter constraint. Most of them are documented in software manuals or service description files.

It is quite common that the input data of different parameters of a specific geoprocessing tool, or the inputs and outputs of different tools within a workflow, are interdependent [7]. For instance, geoprocessing workflows are built upon the output-input dependencies of a sequence of tools. The output data of the preceding tool within a workflow can only be correctly accepted by the following tool if the output data conform to the input parameter constraints of the following tool. Constraints on the interdependent relations of two specific targets are called inter-parameter constraints in this paper. Inter-parameter constraints consist of two subtypes according to the types of relations between parameter inputs. The first is equivalent constraints which require the properties, such as the data theme and coordinate reference system (CRS), of the input/output data of two parameters to be semantically equal to each other. This type of constraint is frequently used in many tasks, such as the composition of geoprocessing workflows. The second subtype is dependent constraints that the validity of the input data of one parameter (the dependent parameter) is dependent on the input data of another parameter (the independent parameter). For example, in the ArcGIS® clip analysis tool, the permissible geometry types of input data for the parameter "clip features" are dependent on the geometry type of input data for the parameter "input features". Specifically, if the input features are polygons, the clip features can only be polygons. Inter-parameter constraints are closely related to individual-level constraints that the latter must be satisfied first.

2.2. Conditions-Oriented Classification of Parameter Constraints

Pre/post-conditions on input data properties are the nucleus of parameter constraints. The properties of a geospatial input dataset can typically be divided into five types (i.e., essential, morphological, provenance, quality, and application-context-matching properties) based on the roles they play in the lifecycle of data (e.g., acquisition, processing, and application) [8,31,32]. Parameter constraints can thus be classified into five subtypes and are highlighted in Table 3.

Table 3. Classification of parameter constraints according to the constrained types of data properties.

Constraint Types (Based on Data Properties)	Description
Essential constraints	Constraints on essential properties such as data theme and spatial/temporal coverage that distinguish the dataset from others.
Morphological constraints	Constraints on morphological properties that describe the internal structure and external shape of the data, such as CRS and data format.
Provenance constraints	Constraints on provenance information that indicate where and how the data are collected and derived from, such as data sources, processing algorithms and steps, etc. They are important to ensure the usability and reliability of input data [33,34].
Quality constraints	Constraints on data quality attributes such as outliers, coverage completeness, accuracy, and results of consistencies verified by geographic databases in structural, geometric, and topo-semantic levels [8,35].
Application-context-matching constraints	Constraints on input data or its properties determined by the specific natural or social application context of the geoprocessing tool, such as suitable value range

Note that the application-context-matching constraints are ignored by existing studies. The application context of a geoprocessing (especially spatial analysis) tool normally consists of several characteristics, including but not limited to the application's purpose (e.g., drainage network delineation), the geographic characteristics of the study area (such as the underlying surface type, terrain conditions, and spatial scale) [31,36,37], and sometimes social conditions such as national laws [38]. It determines the suitability or correctness of parameter inputs and further influences the efficiency of geoprocessing and/or the

reliability of the results [12,36–39]. For example, the resolution of digital elevation models (DEMs) should match both the spatial scale and purpose of hydrological and ecological applications [40]. Application-context-matching constraints are mainly derived from either years of experience, or a theoretical understanding of the geospatial data, or the methodologies/algorithms implemented by the geoprocessing tools. Consequently, it is difficult for end-users of geoprocessing tools, especially non-experts, to master the application-context-matching constraints and to prepare fit-for-application-context input data for geoprocessing tools. Therefore, to ensure the correct functioning of geoprocessing tools and the reliability of the processing results, it is crucial in intelligent geoprocessing to formalize the application-context-matching constraints and to provide useful feedback messages. It is worth noting that not all input data that violate the application-context-matching constraints should be viewed as invalid. Some of them are only inappropriate in the application context because they only affect the accuracy of the processing results rather than the correct functioning of the tool.

3. SHACL-Based Formalization of Parameter Constraints

3.1. Basic Idea and the High-Level RDF Constraint Language-SHACL

In recent years, high-level RDF constraint languages have become an emerging way to define formal constraints over RDF data and validate their conformance [25,41–43]. These languages are designed based on the assumptions of CWA and UNA. Besides, they employ high-level RDF vocabulary to represent abstract constraint concepts, e.g., "sh:minCount" for the minimum cardinality constraint. This makes high-level constraint languages comparatively easy to understand, and users can formalize constraints more intuitively and concisely [44].

Among existing high-level RDF constraint languages, SHACL [27] was declared as a World Wide Web Consortium (W3C) Recommendation (i.e., standard) in 2017. Constraints formalized using SHACL are RDF graphs called "shape graphs" or "shapes"; the input data (in the form of RDF graphs too) being validated is called "data graphs". SHACL primarily provides three key features to formalize constraints. The first is target declarations which allow users to specify the targets that a shape should validate against. The second feature includes two types of constraint components designed for the description of shapes, i.e., the SHACL Core and SHACL-SPARQL. The former provides a set of high-level and standard built-in components to describe the commonly used types of constraints, such as the value range and cardinality. The latter extends the SHACL Core by enabling users to write SPARQL queries to define custom and complex constraints. The third key feature is the properties that can be used to declare the severity and user-friendly messages of shapes. This could help end-users of the tools to easily understand the violations and then correct them.

The three key features of SHACL closely match the three requisite parts of different types of parameter constraints (see Section 2 and Figure 3). This means SHACL is easier to understand and use for formalizing parameter constraints of geoprocessing tools, as compared to SPARQL and its subsets, which are currently used in existing query-oriented methods. The difficulty to define SHACL shapes can be further reduced by some user-friendly tools such as the TopBraid Composer®. Meanwhile, being a standard constraint language means that the constraints formalized using SHACL (i.e., shapes) can be easily shared and reused to reduce repetitiveness. Moreover, SHACL is more flexible than OWL/SWRL and RDF/RDFS [45], thus SHACL-based methods are better than the methods that express and validate constraints based on the extension of these languages or description logics [17,20,21]. Therefore, it is reasonable to use SHACL as the language to formalize different types of parameter constraints.

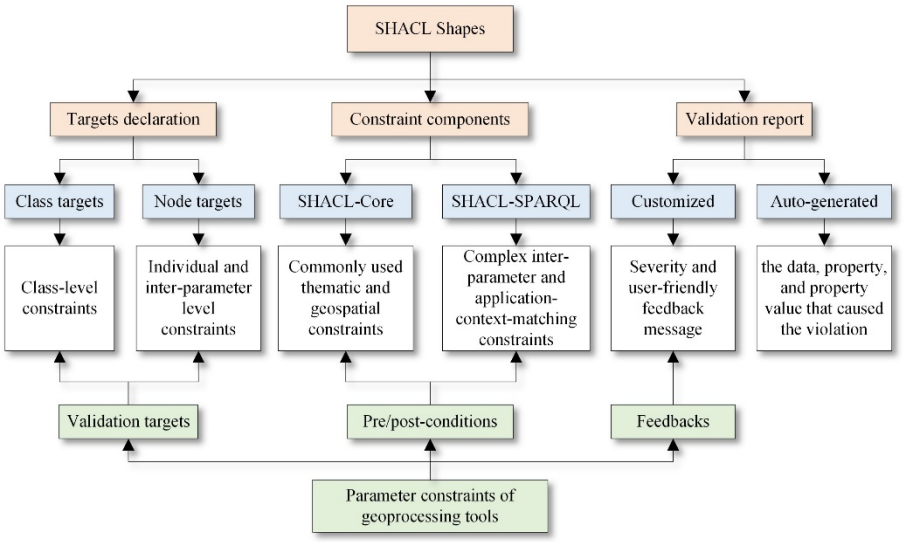

Figure 3. SHACL features and their match to parameter constraints.

Therefore, a SHACL-based method could be designed, with the guidance of the conceptualization and classification of parameter constraints introduced in Section 2, to reduce the difficulty and to improve the efficiency of the formalization of parameter constraints for users, including knowledge engineers and geoprocessing tool developers.

3.2. Overall Design of the Proposed Method

The overall design of the proposed method includes two main stages (Figure 4).

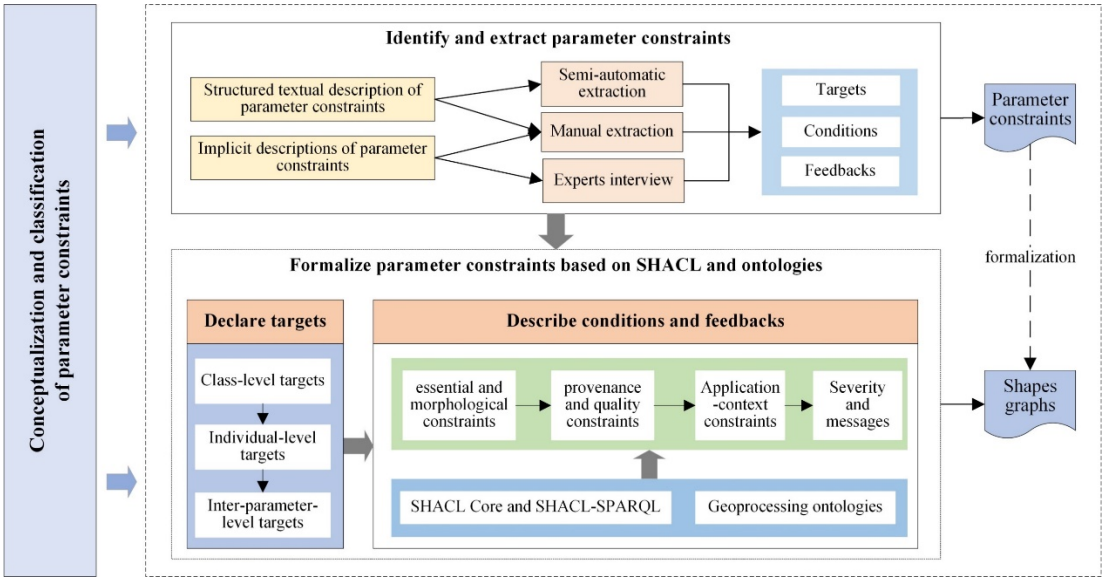

Figure 4. Framework of the proposed method.

The first part of the design of the proposed method is to design a workflow to help users to identify and acquire all the possible types and contents of parameter constraints from proper knowledge sources (such as tool documents and domain experts), which is normally a trivial and cumbersome process.

The second part is to formalize the acquired parameter constraints based on SHACL and geoprocessing ontologies. This part includes the declaration of validation targets, and then the descriptions of constraint conditions and the correspondence feedback based on SHACL Core and SHACL-SPARQL. Domain-specific ontologies are employed to explicitly

describe the semantics of geoprocessing tools and geospatial data within the definitions of parameter constraints. A detailed design of the two parts of the proposed method is presented in the following sub-sections.

3.3. Identification and Acquisition of Parameter Constraints

The first step in the formalization of parameter constraints is to identify and acquire/extract all the possible constraints from proper knowledge sources of a geoprocessing tool. Knowledge engineers could face many challenges during this process, including:

- How many types of constraints could a specific parameter have?
- For each type of parameter constraint, from where could their contents be obtained?
- How do we identify the targets, conditions, and feedback of the constraints?
- How do we acquire the constraints completely and efficiently?

The aforementioned analysis of parameter constraints (Section 2) lays the foundation for knowledge engineers to face these challenges. First, the classification of parameter constraints can be used as a checklist to identify the possible types of constraints that a parameter could have. Second, according to the characteristics of each constraint type and their existing forms, there are two primary types of knowledge sources. Typically, most of the essential and morphological constraints exist in the form of textual descriptions since they are straightforward and can be explicitly documented. Thus, the first type of knowledge source is (semi-)structured software manuals or service description files. The parameter constraints can be extracted from these files manually or by automated information extraction techniques such as web spiders. Meanwhile, many of the domain-specific class-level constraints (e.g., constraints on CRS), along with provenance, quality, and application-context-matching constraints of a parameter, exist in the form of human expertise. Thus, the second type of knowledge source is domain experts and unstructured literature recording such knowledge (e.g., standards and journal papers). In this situation, interviewing domain experts is a reasonable way to identify and acquire the corresponding parameter constraints.

This paper proposes a two-phase process to facilitate the identification and acquisition of parameter constraints (Figure 5). Generally, text extraction is much easier than expert interviews which are typically time-consuming and challenging due to the knowledge divide between knowledge engineers and experts. Particularly for those (semi-)structured description files, e.g., the Extensible Markup Language (XML)-based process profiles, the parameter constraints can be extracted automatically using specifically developed tools or scripts. Thus, the first phase identifies and extracts textual descriptions of parameter constraints from software manuals and/or service description files. Firstly, software-specific class-level constraints (i.e., class-level constraints that are only applicable to the geoprocessing tools of a specific software package) should be extracted to reduce repetitiveness. Then, for each parameter of a geoprocessing tool, its essential and morphological constraints can be identified and extracted directly from the documents according to their characteristics (see Section 2). After that, the inter-parameter constraints which are normally defined based upon individual-level constraints could be identified and extracted.

The second phase identifies and acquires domain-specific class-level constraints, as well as provenance, quality, and application-context-matching constraints of a parameter from geoprocessing experts through interviews. To begin with, constraints extracted in the first phase should be checked by experts to familiarize them with the parameter constraints and to determine whether the extracted constraints of the targeted parameter are complete or not. Afterward, knowledge engineers should ask experts a series of questions to acquire explicit definitions of the constraints of the target parameter, including the conditions and feedback. In particular, application-context-matching constraints should be the last to be acquired. This is because it could be very difficult to explicitly identify and describe the application context (including the task/purpose and study area characteristics) and the corresponding constraints. The interviews could consist of many iterative sessions. The results should be reviewed by different experts to improve the quality.

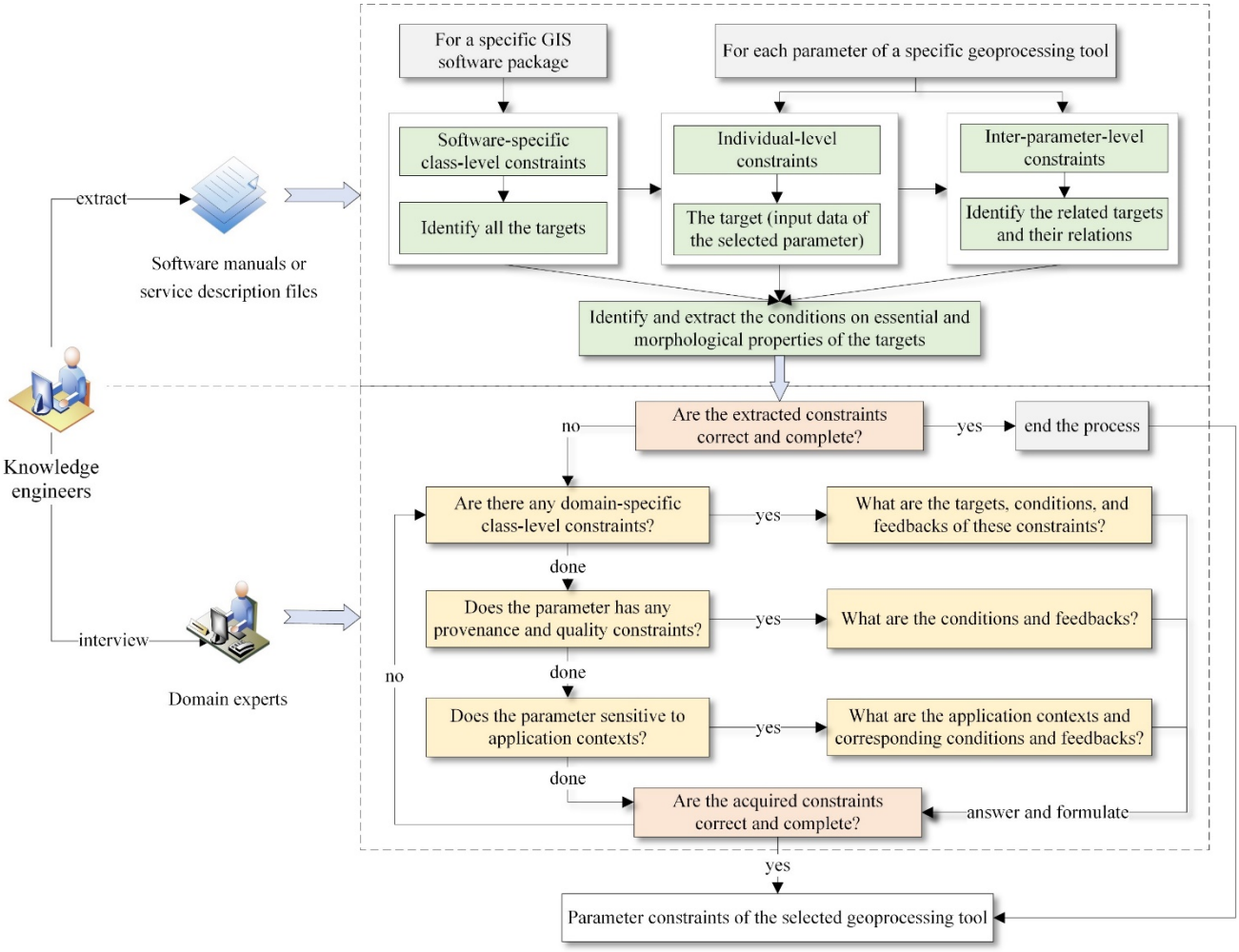

Figure 5. A two-phase process to identify and extract parameter constraints.

3.4. Formalizing Parameter Constraints Based on SHACL and Ontologies

The above-acquired parameter constraints of geoprocessing tools need to be formalized as shapes based on SHACL and geoprocessing ontologies, with the guidance of the classification of parameters constraints described in Section 2. Due to the variety and complexity of parameter constraints, multiple types of knowledge (Figure 6), especially those about geoprocessing tools and geospatial data, are required to support the formalization [1–3,7,46]. For example, general knowledge and geospatial domain-specific knowledge are required to express the constraints on unit and data theme, respectively. These types of knowledge are typically formalized as concepts, properties, and relations in a set of ontologies. For instance, in the process ontology, the predicate process:hasData is defined to describe the relationship between a parameter and its input data. Currently, there are plenty of publicly available ontologies about these types of knowledge, such as the Simple Knowledge Organization System (SKOS) [47] for general metadata, Geooperator [3,48] for geoprocessing tools, GeoSPARQL [49] for geometry types and spatial relations, and GeoDataOnto [32] for geospatial data properties. These existing ontologies have been reused in this paper to support the formalization of parameter constraints.

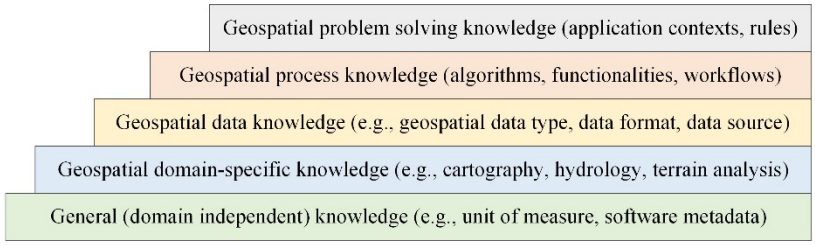

Figure 6. Knowledge required for the semantic description of parameter constraints.

The overall process of the SHACL-based formalization of parameter constraints can be divided into two stages: (1) the declaration of validation target(s), and (2) the formalization of constraint conditions and feedback. In each stage, the following two principles are adopted to simplify formalization and to reduce repetitiveness.

- Reusable constraints should be formalized at the very beginning of the stage. This includes not only the class-level constraints, but also constraints on commonly-used data properties (e.g., the map projection) that exist in many parameters.
- Constraints that can be easily described using the SHACL Core should be formalized before those using SHACL-SPARQL. As SHACL-SPARQL is a trade-off between usability and flexibility, it is comparatively more difficult to understand, write, and maintain than SHACL Core. SHACL-SPARQL is only suitable for complex constraints such as the inter-parameter and application-context-matching constraints.

3.4.1. Target Declarations of Parameter Constraints

Targets of parameter constraints in SHACL are RDF nodes representing the input data of a parameter in the data graphs that a shape should validate against. SHACL-based target declaration represents the relationship between data nodes and shapes as RDF triples which consist of three components, i.e., subject, predicate/property, and object. Specifically, these target declaration triples take the shape as the subject, the pre-defined SHACL properties (e.g., sh:targetNode) as the predicate, and the target(s) as the object. As analyzed in Section 2, there are three types of validation targets that need to be declared: class-level targets, individual-level targets, and inter-parameter level targets.

According to the two aforementioned principles, class-level targets should be declared first. Class-level targets can either be declared explicitly using SHACL predicates, including sh:targetClass, sh:targetSubjectsOf, and sh:targetObjectsOf (e.g., No. 1–3 in Table 4), or implicitly by declaring the shape itself as an ontology class (e.g., No. 4 in Table 4). All the instances of the declared class, or all the subjects/objects of the declared predicate, can be validated directly against the resulting shape.

Table 4. Examples of class-level target declarations.

No.	Example Shapes	Description
1	data:RasterDataShape a sh:NodeShape; sh:targetClass data:RasterData;	declare data:RasterData as the target class explicitly
2	data:RasterData a sh:NodeShape, rdfs:Class.	declare data:RasterData as the target class implicitly
3	data:ParameterShape a sh:NodeShape; sh:targetSubjectsOf process:hasData;	declare the subjects (parameters) of predicate process:hasData as the targets
4	data:DataShape a sh:NodeShape; sh:targetObjectsOf process:hasData;	declare the objects (input data) of predicate process:hasData as the targets

Note: RDF triples are expressed in the Turtle syntax. The "data", "sh", "process", and "rdfs" are prefixes of the namespace of data ontology, SHACL vocabulary, geoprocessing tool ontology, and RDF schema, respectively.

The next step is to declare the targets of the individual-level constraints which have two subtypes: parameter-level and data-level constraints. As one parameter (either input, output, or option) might have two or more input data items, parameter-level targets should be declared first. As shown in Table 1, the target of parameter-level constraints

(called parameter shapes hereafter) is the parameter itself declared using the predicate sh:targetNode (e.g., Line 3 in Table 1). Then, constraints (Line 6) on the input data as a whole (instead of its properties) of the declared parameter are defined as property shapes (begin with Line 4). What needs to be declared next is the targets of data-level constraints which are concrete inputs provided by the end-users of the geoprocessing tool. Instead of declaring the data-level targets explicitly using sh:targetNode, the proposed method declares them implicitly as objects of the predicate process:hasData (or process:hasLiteralData for literal inputs) (Line 5 in Table 1). This is because the predicate sh:targetNode needs its objects to have explicit IRIs (International Resource Identifiers), which is meaningless for data-level targets (i.e., concrete inputs) whose identifier and occurrence number are dynamic.

Listing 1. Declarations of individual-level targets, which is an example of the parameter in_surface_raster of the Flow Direction tool of ArcGIS®.

```
1.  # the parameter shape
2.  arcgis:InSurfaceRasterShape a sh:NodeShape;
3.      # declare the parameter as the target
4.      sh:targetNode arcgis:in_surface_raster;
5.      sh:property [
6.          sh:path process:hasData;
7.          sh:class data:RasterData; # the class-level constraints
8.          sh:minCount 1; sh:maxCount 1;
9.          # the data shape
10.         sh:property [
11.             # data theme condition
12.     sh:path [sh:alternativePath (dcterms:subject dcat:theme)];
13.     sh:minCount 1;
14.     sh:in (vocab:filled_DEM vocab:hydrologically-corrected_DEM);
15.         ];
16. ].
```

The validation of inter-parameter constraints needs to compare the input data properties of two or more interdependent parameters. However, this is beyond the capabilities of the built-in target declaration predicates such as sh:targetNode, because SHACL Core does not currently provide the appropriate vocabulary for the declaration of inter-parameter-level targets. Thus, SHACL-SPARQL is used by the proposed method to declare inter-parameter targets in two steps. The first step is to declare the primary parameter as the target (Line 3 in Table 2), which is the same as the declaration of parameter-level targets. The second step is to write SPARQL queries to select the data-level targets (the input data, Lines 9 and 10 in Table 2) that needed to be constrained from the target and the dependent parameter (the arcgis:in_features in Table 2). Then, conditions of the inter-parameter constraints could be represented by querying and comparing the property values of the selected data-level targets. The details of the expression of constraint conditions will be described in the next section.

3.4.2. Formalization of Constraint Conditions and Feedback

Once the validation targets have been declared, the conditions and violation feedback on input data properties of these targets could be formalized as node or property shapes based on SHACL and geoprocessing ontologies. We mainly focus on the formalization of conditions because the description of feedback is relatively straightforward.

Listing 2. Declaration of inter-parameter targets based on SHACL-SPARQL with an example of the parameters in_features and clip_features of the clip tool of ArcGIS®.

```
arcgis:clipFeaturesShape a sh:NodeShape;
# step1: declare the primary parameter as the target
    sh:targetNode arcgis:clip_features;
    sh:sparql [
      sh:select """
      SELECT $this (dcterms:type AS ?path) (?clip_geom_type AS ?value)
      WHERE {
# step 2: select data-level targets, i.e., ?clip_features_data and ?in_features_data
          $this process:hasData ?clip_features_data.
          arcgis:in_features process:hasData ?in_features_data.
          # select the data properties that need to be constrained
          ?clip_features_data dcterms:type ?clip_geom_type.
          ?in_features_data dcterms:type ?in_geom_type.
          # compare the selected properties (omitted for brevity)
        } """;
].
```

The formalization of conditions could be divided into four steps. The first step is to formalize the frequently used conditions of the values (either literal values or complex objects represented as RDF graph nodes) of a common data property as reusable shapes. These shapes can be further referenced by others to define more specific and complex shapes so as to save time and effort. For instance, many spatial analysis tools require the input data to be projected to a map projection. This means the corresponding conditions would be formalized many times among different tools. Thus, to avoid repetition, we formalize the conditions as a reusable property shape (e.g., the data:ProjectedShape illustrated in Table 3). Any other parameters that need the input data to be projected could reuse it in the definition of the corresponding parameter shape, e.g., Line 15 in Table 3.

Listing 3. Formalizing commonly used conditions as reusable shapes, which is an example of the shape for the projected spatial data.

```
data:ProjectedShape a sh:PropertyShape;
    sh:path data:isProjected;
    sh:minCount 1; sh:maxCount 1; # cardinality constraints
    sh:datatype xsd:boolean;
    sh:hasValue true;
    # the feedback message for violations
    sh:message "The input data must be projected"@en.

# the parameter shape defined in Table 1
arcgis:InSurfaceRasterShape a sh:NodeShape;
      sh:targetNode arcgis:in_surface_raster;
      sh:property [
          sh:path process:hasData;
          sh:property data:ProjectedShape; # reuse the property shape
          # other conditions (omitted for brevity)
].
```

The second step is to formalize the acquired conditions of the essential, morphological, provenance, and quality properties of a given target as property shapes. The data property that needs to be constrained is specified as the object of the predicate sh:path (Line 2 in Table 3). The vocabulary for describing the data properties (e.g., data:isProjected) is defined in the data ontology. SHACL Core constraint components are used to describe the corresponding conditions (e.g., the sh:minCount at Line 3 in Table 3 for the minimum cardinality constraint).

The third step is to formalize the conditions of inter-parameter constraints by writing SPARQL queries based on SHACL-SPARQL. Generally, these queries are written in a combination with the query statements defined for the declaration of inter-parameter targets. This step first writes queries to select the values of the data properties that need to be constrained (Line 8 and 9 in Table 4). Then, filter expressions (Line 12 in Table 4) are defined to compare the selected property values with each other. The comparative result determines the conformity of the two inputs. The input data of the dependent parameter will be viewed as invalid if the result value of the filter is "true". For example, as shown in Table 4, the query statement will return "true" if the input data of parameter arcgis:in_features do not subsume the input data of the dependent parameter arcgis:clip_features. Note that logical reasoning must be performed before the validation in this case to deduce the semantic relation (the data:subsumesGeometry) between the two inputs.

Listing 4. Formalizing inter-parameter conditions based on SHACL-SPARQL, which is an example of the parameters in_features and clip_features of the clip tool of ArcGIS®.

```
1.  arcgis:clipFeaturesShape a sh:NodeShape;
2.      sh:targetNode arcgis:clip_features;
3.      sh:sparql [
4.        sh:select """
5.        SELECT $this (dcterms:type AS ?path) (?clip_geom_type AS ?value)
6.        WHERE {
7.          # simplify the query use SPARQL property paths
8.          $this process:hasData/dcterms:type ?clip_geom_type.
9.          arcgis:in_features process:hasData/dcterms:type ?in_geom_type.
10.         # filter out the input data that does not conform.
11.          FILTER NOT EXISTS
12.                  {?in_geom_type data:subsumesGeometry ?clip_geom_type}
13.       }""";
14. ].
```

The last step is to formalize the application-context-matching conditions based on SHACL-SPARQL. This step could be complex since the applicable contexts of a tool may include not only the application purpose, but also the geographic characteristics (such as the spatial scale) of the study area. Thus, the proposed method divides this step into two sub-steps. The first sub-step is to write SPARQL queries to select the expected characteristics of the real-world application context and the properties of the targeted input data that need to be constrained. The second sub-step is to write filter expressions to compare the expected application context characteristics with those of the real-world application, and to compare the conditions with the literal value or input data properties.

Table 4 illustrates such a shape which constrains the input value of the parameter "flow_direction_type" of the Flow Direction tool in ArcGIS® 10.7. This parameter determines which flow direction algorithm should be used to generate the results. The shape here describes that the value should be "MFD" (short for Multiple Flow Direction) if the application purpose (Line 14) of the tool (Line 11) is to calculate the spatial pattern of hydrological parameters, such as the flow accumulation and topographic wetness index [50–52].

The detailed design of the proposed method presented above elaborated on how to identify different types of parameter constraints and how to further formalize their contents based on SHACL and ontologies. Such a method could free users from the burden of acquiring all the possible types of parameter constraints and formalizing them with verbose SPARQL queries.

Listing 5. Formalization of application-context-matching conditions, which is an example of the Flow Direction tool in calculating the spatial pattern of hydrological parameters.

```
arcgis:FlowDirectionApplicationContextShape a sh:PropertyShape;
      sh:path process:hasLiteralData;
      sh:severity sh:Warning;    # report a warning instead of violation
      sh:sparql [
         sh:message "The MFD algorithm is better than D8 and DINF algorithms for calculating the spatial pattern of hydrological parameters."@en;
         sh:select """
              SELECT $this ?value
              WHERE {
                   # define the application context
                   arcgis:flow_direction_tool context:applicationContext ?a;
                            process:hasInput          $this.
                   # select the application purpose
                   ?a    context:applicationPurpose    ?purpose.
                   # select the (literal) input data: the validation target
                   $this process:hasLiteralData                ?value.
             # filter the data according to the condition and application context
             FILTER(IF((?purpose a context:CalHydroParamsSpatialPattern) && (UCASE(STR(?value))="MFD"), false, true))}
       """].

arcgis:FlowDirectionTypeShape a sh:NodeShape;
sh:targetNode arcgis:flow_direction_type;
# reuse the shape
sh:property arcgis:FlowDirectionApplicationContextShape;
.
```

Note: The context:CalHydroParamsSpatialPattern is an ontology class that represents all the purposes of calculating the spatial pattern of hydrological parameters, such as the flow accumulation and topographic wetness index.

4. Application Case

This paper provides a case study (i.e., the Flow Direction tool in ArcGIS® 10.7 and its parameter constraints) to illustrate the application of the proposed SHACL-based formalization method for parameter constraints. The emphasis here is on how the proposed method can be used to formalize the parameter constraints of a geoprocessing tool, and the effectiveness of the formalization results (i.e., shapes) in the validation of parameter inputs.

4.1. Case Design

4.1.1. The Flow Direction Tool and Its Parameter Constraints

The Flow Direction tool is designed to calculate the flow directions of a study area based on digital elevation model (DEM) and flow direction algorithms. It has five parameters, including one mandatory input, one optional output, and three optional options. To keep things simple, this case focuses only on the constraints of the input parameter "in_surface_raster" and the "flow_direction_type" option (Table 5). It is worth noting that filling the depressions in DEMs is a necessary pre-processing step for flow direction calculation, so as to derive hydrologically correct (or filled) DEMs [53]. Therefore, the input data of the parameter "in_surface_raster" should be pre-processed by depression-processing tools such as the Fill tool of ArcGIS®. Moreover, its data theme should be semantically equal to "Filled-DEM", rather than the original "DEM".

4.1.2. Extraction and Formalization of Parameter Constraints of the Flow Direction Tool

Following the two steps of extracting the parameter constraints for using the proposed method (Section 3.3), we first manually extracted the explicitly described constraints from the reference web page of the tool (https://desktop.arcgis.com/en/arcmap/10.7/tools/spatial-analyst-toolbox/flow-direction.htm, accessed on 5 April 2021), including the data theme, data/value type, value range, and cardinality constraints. The second step acquired

the other constraints (see Table 5) from domain-specific literature and experts via manual extraction and interviews, respectively. The application-context-matching-constraint on the "flow_direction_type" option is extracted from academic publications [50–52] discussing flow direction algorithms and their applications.

Table 5. Constraints of the parameter "in_surface_raster" and "flow_direction_type".

Parameter	Constraint Types and Properties	Constraint Conditions
in_surface_raster	Essential: data theme	semantically equal to Filled-DEM or Hydrologically-corrected DEM
	Morphological: cardinality	only 1
	Morphological: value type	string
	Morphological: data type	grid raster
	Morphological: CRS	must be projected
	Provenance: pre-processing tool	should be arcgis:Fill
flow_direction_type	Morphological: cardinality	at most 1
	Morphological: value type	string
	Morphological: value range	only D8, MFD, Dinf are allowed
	Application-context-matching constraints	see Table 5

Following the formalization principles and steps described in Section 3.4, the extracted constraints were formalized as shapes based on SHACL and geoprocessing ontologies. Table 1, Table 3, and Table 5 presented in Section 3.4 illustrated the shapes of the core constraints of the input parameter "in_surface_raster"; listing 2 in Section 3.4 described the shapes of the core constraints of the "flow_direction_type" option. Details of these shapes could be found via the link presented in the Section "Data Availability Statement".

4.1.3. Application Context and Input Data of the Tool

To evaluate the effectiveness of the formalized parameter constraints (especially the application-context-matching constraints for the flow direction type), we assumed the application purpose of this case was to calculate the topographic wetness index (TWI) of a small watershed within Fujian Province, China (Figure 7a), based on a gridded DEM. Flow direction calculation, based on the Flow Direction tool, is a necessary step in the TWI calculation workflow (Figure 7b). The flow direction type used in this case is assumed to be "D8" which is a single flow direction algorithm and is unsuitable for the given application purpose. Additionally, we assumed that the DEM dataset had not been pre-processed by the Fill tool, and its coordinate system was World Geodetic System 1984 (WGS84). This means the DEM did not have the expected data theme and CRS.

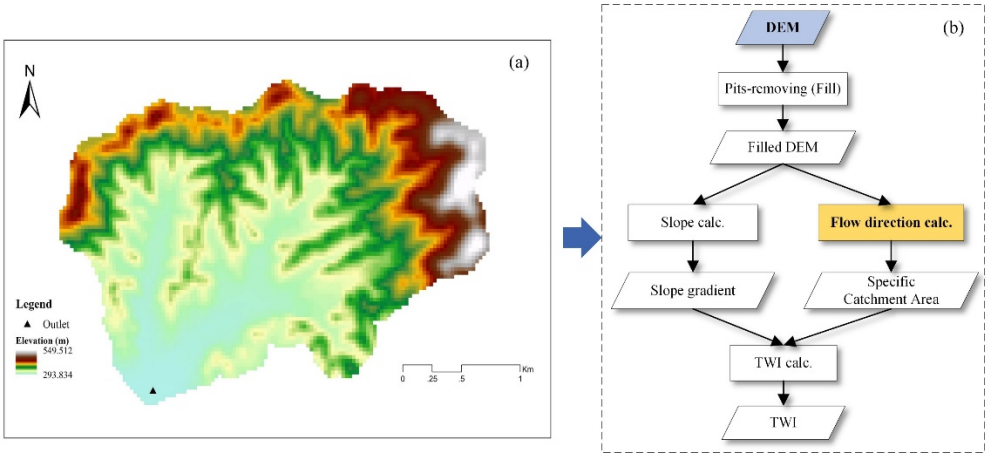

Figure 7. DEM of a small watershed (**a**) and the TWI calculation workflow (**b**) for the application case.

The abovementioned application context and input data of the two selected parameters have been automatically transformed into data graphs (Table 6). The Python scripts developed for the transformation are based on Python, GDAL (Geospatial Data Abstraction Library) (https://gdal.org/, accessed on 5 April 2021), and RDFLib (https://github.com/RDFLib/rdflib, accessed on 5 April 2021). The corresponding scripts and data graphs have been shared together with the shapes in the repository whose link can be found in the Section "Data Availability Statement".

Listing 6. RDF graphs of the application context and input data used in this case.

```
# declare the parameters and application context
arcgis:flow_direction_tool a arcgis:SpatialAnalystTool;
    process:hasInput arcgis:flow_direction_type, arcgis:in_surface_raster;
    context:applicationContext [
        context:applicationPurpose context:topographic_index_calculation.
    ].
# parameters and input data
arcgis:flow_direction_type
    dcterms:identifier "flow_direction_type";
    process:hasLiteralData "D8".

arcgis:in_surface_raster
    dcterms:identifier "in_surface_raster";
    process:hasData [
        a data:RasterData;
        data:hasCRS data:WGS84;
        data:isProjected false;
        dcat:theme vocab:DEM. # instead of "filled-DEM"
    ].
```

4.2. SHACL-Based Input Data Validation and the Results

A validation report (Figure 8) was generated by validating the data graphs (Table 6) against the shapes of the parameter constraints formalized using the proposed method. An open-source implementation of the SHACL validator called pySHACL (https://github.com/RDFLib/pySHACL, accessed on 5 April 2021) was used to perform the validation. The conformance of the parameter inputs against the shapes was false, which means the inputs were invalid (the part (1) in Figure 8). Specifically, the two inputs had violated four constraints, namely the data theme, map projection, provenance, and application-context-matching constraints (parts (2)–(5) in Figure 8). The generated result messages (e.g., Line 5 in Figure 8), along with the objects of the predicates sh:resultPath and sh:value (e.g., Line 10 and 14 in Figure 8), explicitly explained the reasons why these inputs were invalid. The messages defined during the formalization also offer user-friendly instructions from which end-users of the tools could easily understand and correct the errors to improve the quality of the input data.

The shapes and validation results of this application case show that the proposed method provides a comparatively effective and flexible way to formalize the parameter constraints. Users are facilitated to identify all the possible parameter constraints from different knowledge sources based on the designed extraction workflow (Section 3.3). Furthermore, the extracted parameter constraints can be correctly and easily formalized by following the detailed designed steps (Section 3.4) and by using the high-level vocabulary and constraint components (SHACL Core and SHACL-SPARQL) of SHACL. Besides, the proposed method considers the application-context-matching constraints which have been ignored by existing studies. As a result, validators of intelligent geoprocessing can not only validate the correctness, consistency, and completeness, but also the appropriateness of the input data to the application context.

```
[] a sh:ValidationReport ;
   sh:conforms false ;                                                    (1)
   sh:result [ a sh:ValidationResult ;
         sh:focusNode _:ub5bL65C21 ;
         sh:resultMessage "The DEM should be filled or removed pits/sinks"@en ;
         sh:resultPath _:ub6bL32C14 ;
         sh:resultSeverity sh:Violation ;
         sh:sourceConstraintComponent sh:InConstraintComponent ;          (2)
         sh:sourceShape data:FilledDEMShape ;
         sh:value <https://gcmdservices.gsfc.nasa.gov/kms/concept/digital_elevation_models> ],
      [ a sh:ValidationResult ;
         sh:focusNode _:ub5bL65C21 ;
         sh:resultMessage "The data must be projected"@en ;
         sh:resultPath data:isProjected ;
         sh:resultSeverity sh:Violation ;
         sh:sourceConstraintComponent sh:HasValueConstraintComponent ;    (3)
         sh:sourceShape data:ProjectedShape ],
      [ a sh:ValidationResult ;
         sh:focusNode _:ub5bL65C21 ;
         sh:resultMessage "The DEM used for calculating flow directions should be filled using tool ArcGIS Fill" ;
         sh:resultPath process:wasGeneratedByTool ;
         sh:resultSeverity sh:Violation ;
         sh:sourceConstraintComponent sh:HasValueConstraintComponent ;    (4)
         sh:sourceShape _:ub6bL52C14 ],
      [ a sh:ValidationResult ;
         sh:focusNode arcgis:flow_direction_type ;
         sh:resultMessage "The MFD algorithm is better than D8 and DINF algorithms when calculating the spatial
         pattern of hydrological parameters such as topographic index."@en;
         sh:resultPath process:hasLiteralData ;
         sh:resultSeverity sh:Warning ;] ;                                (5)
         sh:sourceConstraintComponent sh:SPARQLConstraintComponent ;
         sh:sourceShape arcgis:FlowDirectionApplicationContextShape ;
         sh:value "D8"^^xsd:string ] .
```

Figure 8. Validation report of the input data against the shapes of the Flow Direction tool. Part (1) represents the conformance of the inputs as a whole, parts (2)–(5) represent the validation results of the inputs against the data theme constraint, map projection constraint, provenance constraint, and application-context-matching constraints, respectively.

5. Evaluation and Discussion

5.1. Evaluation Method

Usability refers to the extent to which a method can be easily learned, understood, and used by users (including novices and experts) to accomplish a specific task effectively and efficiently in a specified application context [54,55]. Methods with a high degree of usability could not only reduce the difficulty of formalizing parameter constraints but could also decrease the probability of failure, so as to guarantee the quality of the formalization results. Currently, one of the most recognized categories of usability evaluation methods is the heuristic evaluation and its extensions [55–57].

This paper employs usability as the criterion and the heuristic evaluation as the evaluation method to evaluate the proposed method along with the inference-oriented and query-oriented methods. The existing heuristics are generally used for website usability purposes [54,55], thus are unsuitable to be reused directly in this study. Table 6 lists the heuristics and criteria used for the evaluation in this study. They are designed referencing [54,55] with the consideration of the specificity on formalizing parameter constraints. The explanation contents in Table 6 present the rationale for the selection of heuristics and criteria. Specifically, among these usability heuristics, the correctness, completeness (coverage), error prevention, as well as flexibility and extendibility are designed to evaluate the effectiveness of the methods; the learnability is designed to evaluate the difficulty involved in learning; the other heuristics are adopted for the efficiency evaluation. For example, the standard and consistency of the evaluated method could make it easy to share and reuse the already formalized constraints and could thereby reduce repetitive works.

Table 6. Usability heuristics for the evaluation of parameter constraint formalization methods.

Usability Heuristic	Evaluation Criteria	Explanation
Correctness	(1) the method is designed under CWA and UNA	The ability of the validator to correctly detect all the invalid input data based on the formalized constraints. This is a restrictive criterion.
Completeness (coverage)	(2) number of target-oriented constraints types (3) number of conditions-oriented constraints types	The number of parameter constraint types the method is able to formalize. Only consider the constraints that have been explicitly mentioned in the design of the method under evaluation.
Error prevention	(4) can customize the violation severity (5) can customize end-user-friendly messages	The ability to define the severity and friendly feedback messages to facilitate end-user-understand and deal with input data violations
Flexibility and extendibility	(6) can be used for different tasks (constraint types) (7) support the formalization of new constraint types	The ability to formalize parameter constraints for different tasks in different contexts, including those that have not been pre-defined in the method.
Learnability	(8) provide high-level (abstract) constraint concepts (9) provide listed formalization steps (guidelines) (10) provide help and documentation	Whether the method under evaluation has provided means to facilitate the intended users to learn, understand, and use the method (not only the underlying language).
Standard and consistency	(11) the method is designed based on a constraint formalization standard	Whether the method follows a standard for constraints to ensure the constraints formalized by different users have consistent style and meaning.
Efficiency of use	(12) support the definition of class-level constraints (13) can reuse formalized constraints to define others (14) formalization tools available	The ability to reduce time and efforts expended in the formalization of parameter constraints. The tools must support the formalization of parameter constraints directly, not only general rules or SPARQL queries.

Based on the usability heuristics, the usability of different formalization methods can be evaluated based on the number of usability features (the so-called usability score) that a method could have. The higher the usability score, the better is the corresponding method. Note that the correctness heuristic is a restrictive criterion, on which the methods based on CWA and UNA are shown to be better than others (see Section 1).

5.2. Evaluation Results and Discussion

Table 7 lists the evaluation results of the proposed SHACL-based method and the two existing categories of formalization methods.

Table 7. Results of the usability evaluation of the formalization methods.

Usability Heuristic	Evaluation Criteria *	Features Count of the Formalization Methods		
		Inference-Oriented Methods	SPARQL-Based Methods	The Proposed SHACL-Based Method
correctness	(1)	■ 0	■ 1	■ 1
Completeness (coverage)	(2)	■ 3	■ 2	■ 3
	(3)	■ 3	■ 1	■ 5
Error prevention	(4)	■ 0	■ 0	■ 1
	(5)	■ 0	■ 1	■ 1
Flexibility and extendibility	(6)	■ 1	■ 1	■ 1
	(7)	■ 0	■ 1	■ 1
	(8)	■ 1	■ 0	■ 1
Learnability	(9)	■ 0	■ 0	■ 1
	(10)	■ 1	■ 0	■ 1
Standard and consistency	(11)	■ 0	■ 0	■ 1
	(12)	■ 1	■ 1	■ 1
Efficiency of use	(13)	■ 1	■ 1	■ 1
	(14)	■ 1	■ 0	■ 1
Usability score		12	9	20

* Note: See Table 6 for the detailed definitions of the evaluation criteria.

The proposed method has a larger usability score than the existing methods (Table 7), which means that the proposed method could achieve a better performance in the formalization of parameter constraints. The evaluation results of the completeness (coverage) show that the proposed method comparatively covers more types of parameter constraints. Thus, the proposed method can detect and further refine more potential violations that exist in input data, and so ensures the correct functioning of geoprocessing tools and the reliability of the results. Comparatively, the identification and formalization of parameter constraints in the existing methods are hardly complete or systematic because they lack an explicit classification of parameter constraints.

The results from the learnability criterion show that the proposed SHACL-based method is easier to learn than in previous studies, in particular the SPARQL-based methods. This is primarily due to the fact that users normally are more familiar with high-level languages which provide concise and intuitive constraint concepts. Whereas SPARQL is a low-level language, users (especially the novices) of the SPARQL-based methods must pay more attention to the verbose implementation details (i.e., writing the queries) instead of the formalization of parameter constraints. For example, as illustrated in Table 8, it is even difficult for experienced users to identify the targets and conditions of the constraints formalized using SPARQL, as compared to the shapes defined based on the proposed method. Although SHACL-SPARQL is still as difficult as the original SPARQL, it is only required for complex and less common constraints (such as the inter-parameter constraints). Besides, in contrast to existing studies, this paper and the SHACL community have both offered documents (e.g., the book by Gayo, Prud'hommeaux, Boneva and Kontokostas [19])

and explicit workflows (e.g., Figure 5) for the formalization of constraints. This offers users clear and well-organized ways to identify and formalize parameter constraints.

Table 8. Comparison of constraints formalized based on SPARQL and SHACL.

Method	Example of Formalized Constraints
the SPARQL-based method [22]	geop:Project dt:has_expression "ASK WHERE { geod:in_dataset dt:has_geometry ?g. ?g dt:hasSRS ?SRS. FILTER (?SRS!= ""). }". geop:Project dt:has_message "Your input dataset–(@{geod:in_dataset.rdf:type.?}) does not have a defined coordinate reference system".
the proposed SHACL-based method	dt:SRSShape a sh:NodeShape; sh:targetObjectsOf dt:has_geometry; sh:property [sh:path dt:hasSRS; sh:minLength 1; sh:message "Your input dataset does not have a defined coordinate reference system" @en;].

The evaluation results of the rest of the criteria show that the proposed method is more efficient than others when the SPARQL-based method performs the worst. This is because the proposed method adopted two strategies in its design to improve efficiency. The first strategy is to reduce repetitive efforts in three specific ways. The first way is the adoption of a standard constraint language, i.e., the SHACL, as indicated by the evaluation results of the standard and consistency criterion. This enables users to formalize constraints using standard and reusable components and vocabulary. Consequently, repetitive formalizations could be reduced by reusing and aggregating the shapes (i.e., formalized constraints) shared by the SHACL communities. The second way is the formalization of class-level constraints which can be automatically applied to a group of targets (Table 4). The third way is the shapes reference mechanism by which users can reuse the pre-defined node and property shapes, as illustrated in Table 3. The other strategy is to adopt easy-to-use formalization tools, most of which are ontology editors, such as the Protégé®. These tools could help users by reducing the burdens of recalling the vocabulary and manually writing the constraints.

6. Conclusions and Future Work

Input data validation based on formalized parameter constraints is a key step in intelligent geoprocessing to ensure the correct functioning of geoprocessing tools and the reliability of results. However, how to easily and efficiently formalize parameter constraints is still a problem that is far from being resolved. This paper proposes a new method to identify and formalize parameter constraints based on SHACL, with the support of geoprocessing ontologies and the guidance of the conceptualization and classification of parameter constraints.

An application case based on the ArcGIS® Flow Direction tool was conducted to demonstrate the application and to evaluate the effectiveness of the proposed method in formalizing parameter constraints, such as the application-context-matching constraints which have currently been ignored by researchers.

A usability evaluation was performed to evaluate the proposed method in improving the efficiency and reducing the difficulty in formalizing parameter constraints, as compared to the existing methods. The evaluation results show that the proposed method can formalize more types of parameter constraints in a comparatively easier and more efficient way, even for non-expert users who are unfamiliar with parameter constraints and SHACL.

This means not only the widespread applicability of the proposed method but also the better performance of the proposed method in input data validation in intelligent geoprocessing. It may promote research fields that require constraints formalization over geospatial input data, such as geographic models and integrated modeling environments [58–60].

Further studies are needed to enhance and promote the proposed method. Firstly, automatic methods and tools are needed to reduce users' effort and manual burden in extracting parameter constraints and writing the shapes. Secondly, formalizing complex parameter constraints (e.g., the inter-parameter constraints) based on SHACL-SPARQL is still difficult, as compared to that based on SHACL Core. Thus, to further improve the usability of the proposed method, SHACL Core should be extended to provide intuitive and concise constraint components for the formalization of commonly-used types of complex constraints. Thirdly, the heuristic method used in this paper mainly evaluates the usability of the proposed method from the view of experts [61,62]. The method usability that relates directly to non-expert end-users might be ignored. Thus, the user-centered usability evaluation methods, such as the user testing method which is a compliment of the heuristic method [61,62], should be used in the future to evaluate all aspects of the usability of the proposed method. Last but not the least, violations detected against formalized parameter constraints need to be addressed with proper solutions (e.g., reprojection tools) so that the input data could be transformed to be valid for geoprocessing. Therefore, an important future research issue in intelligent geoprocessing is to develop a new method, combined with the proposed method, to formalize the knowledge that could infer proper solutions for input data violations.

Author Contributions: Conceptualization, Zhi-Wei Hou; Data curation, Zhi-Wei Hou, Yi-Jie Wang, Peng Liang and Yu-Jing Wang; Funding acquisition, Cheng-Zhi Qin; Methodology, Zhi-Wei Hou; Project administration, Cheng-Zhi Qin; Resources, Cheng-Zhi Qin and A-Xing Zhu; Software, Zhi-Wei Hou; Supervision, Cheng-Zhi Qin and A-Xing Zhu; Visualization, Zhi-Wei Hou; Writing—original draft, Zhi-Wei Hou; Writing—review & editing, Cheng-Zhi Qin, Yi-Jie Wang, Peng Liang, Yu-Jing Wang and Yun-Qiang Zhu. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Chinese Academy of Sciences (No. XDA23100503). Supports to A-Xing Zhu through the Vilas Associate Award, the Hammel Faculty Fellow Award, and the Manasse Chair Professorship from the University of Wisconsin-Madison are greatly appreciated.

Data Availability Statement: The data graphs, shapes, and programming code to generate the data graphs and validation reports that support this study's findings are available with the identifier at the link https://figshare.com/s/ae7c8afaee7c939d20aa, accessed on 14 May 2021.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Di, L.; Zhao, P.; Yang, W.; Yue, P. Ontology-driven automatic geospatial-processing modeling based on web-service chaining. In Proceedings of the sixth Annual NASA Earth Science Technology Conference, College Park, MD, USA, 27–29 June 2006; pp. 27–29.
2. Lutz, M. Ontology-Based Descriptions for Semantic Discovery and Composition of Geoprocessing Services. *GeoInformatica* **2007**, *11*, 1–36. [CrossRef]
3. Hofer, B.; Mäs, S.; Brauner, J.; Bernard, L. Towards a knowledge base to support geoprocessing workflow development. *Int. J. Geogr. Inf. Sci.* **2016**, *31*, 694–716. [CrossRef]
4. Scheider, S.; Nyamsuren, E.; Kruiger, H.; Xu, H. Geo-analytical question-answering with GIS. *Int. J. Digit. Earth* **2020**, *14*, 1–14. [CrossRef]
5. Sudmanns, M.; Tiede, D.; Lang, S.; Baraldi, A. Semantic and syntactic interoperability in online processing of big Earth observation data. *Int. J. Digit. Earth* **2017**, *11*, 95–112. [CrossRef]
6. Kruiger, H.; Kasalica, V.; Meerlo, R.; Lamprecht, A.L.; Simon, S. Loose programming of GIS workflows with geo-analytical concepts. *Trans. GIS* **2020**, *25*, 424–449. [CrossRef]
7. Fitzner, D. Formalizing Cross-Parameter Conditions for Geoprocessing Service Chain Validation. In *Emerging Methods and Multidisciplinary Applications in Geospatial Research*; IGI Global: Hershey, PA, USA, 2011; pp. 282–300.
8. Cruz, S.A.B.; Monteiro, A.M.V.; Santos, R. Automated geospatial Web Services composition based on geodata quality requirements. *Comput. Geosci.* **2012**, *47*, 60–74. [CrossRef]

9. Qi, K.; Gui, Z.; Li, Z.; Guo, W.; Wu, H.; Gong, J. An extension mechanism to verify, constrain and enhance geoprocessing workflows invocation. *Trans. GIS* **2015**, *20*, 240–258. [CrossRef]
10. Wiemann, S.; Karrasch, P.; Bernard, L. Ad-hoc combination and analysis of heterogeneous and distributed spatial data for environmental monitoring—Design and prototype of a web-based solution. *Int. J. Digit. Earth* **2018**, *11*, 79–94. [CrossRef]
11. Hofer, B.; Brauner, J.; Jackson, M.; Granell, C.; Rodrigues, A.; Nüst, D.; Wiemann, S. Descriptions of Spatial Operations—Recent Approaches and Community Feedback. *Int. J. Spat. Data Infrastruct. Res.* **2015**, *10*, 124–137.
12. Hou, Z.-W.; Qin, C.-Z.; Zhu, A.-X.; Liang, P.; Wang, Y.-J.; Zhu, Y.-Q. From Manual to Intelligent: A Review of Input Data Preparation Methods for Geographic Modeling. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 376. [CrossRef]
13. Martin, D.; Burstein, M.; Hobbs, J.; Lassila, O.; McDermott, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T. OWL-S: Semantic markup for web services. *W3C Memb. Submiss.* **2004**, *22*, 2004–2007.
14. Roman, D.; Keller, U.; Lausen, H.; Bruijn, J.D.; Stollberg, M.; Polleres, A.; Feier, C.; Bussler, C.; Fensel, D. Web Service Modeling Ontology. *Appl. Ontol.* **2005**, *1*, 77–106.
15. Lutz, M.; Lucchi, R.; Friis-Christensen, A.; Ostländer, N. A rule-based description framework for the composition of geographic information services. In Proceedings of the International Conference on GeoSpatial Sematics, Mexico City, Mexico, 29–30 November 2007; pp. 114–127.
16. Xing, H.; Chen, J.; Wu, H.; Hou, D. A Web Service-Oriented Geoprocessing System for Supporting Intelligent Land Cover Change Detection. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 50. [CrossRef]
17. Tao, J.; Sirin, E.; Bao, J.; McGuinness, D.L. Integrity Constraints in OWL. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, GA, USA, 11–15 July 2010.
18. Bosch, T.; Acar, E.; Nolle, A.; Eckert, K. The role of reasoning for rdf validation. In Proceedings of the 11th International Conference on Semantic Systems, SEMANTICS '15, New York, NY, USA, 16 September 2015; pp. 33–40.
19. Gayo, J.E.L.; Prud'hommeaux, E.; Boneva, I.; Kontokostas, D. *Validating RDF Data*; Morgan & Claypool: San Rafael, CA, USA, 2018; Volume 7.
20. Patel-Schneider, P. Using Description Logics for RDF Constraint Checking and Closed-World Recognition. In Proceedings of the AAAI15: Twenty-Ninth Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015.
21. Shu, Y. A Practical Approach to Modelling and Validating Integrity Constraints in the Semantic Web. *Knowl.-Based Syst.* **2018**, *153*, 29–39. [CrossRef]
22. Hofer, B.; Papadakis, E.; Mäs, S. Coupling Knowledge with GIS Operations: The Benefits of Extended Operation Descriptions. *ISPRS Int. J. Geo-Inf.* **2017**, *6*, 40. [CrossRef]
23. Scheider, S.; Ballatore, A.; Lemmens, R. Finding and sharing GIS methods based on the questions they answer. *Int. J. Digit. Earth* **2018**, *12*, 594–613. [CrossRef]
24. Harris, S.; Seaborne, A. SPARQL 1.1 Query Language. Available online: https://www.w3.org/TR/sparql11-query/ (accessed on 13 December 2019).
25. Bosch, T.; Eckert, K. Requirements on RDF Constraint Formulation and Validation. In Proceedings of the the 14th DCMI InternationalConference on Dublin Core and Metadata Applications (DC 2014), Austin, TX, USA, 12–15 October 2014; pp. 95–108.
26. Kontokostas, D.; Westphal, P.; Auer, S.; Hellmann, S.; Lehmann, J.; Cornelissen, R.; Zaveri, A. Test-driven evaluation of linked data quality. In Proceedings of the the 23rd International Conference on World Wide Web, Seoul, Korea, 7–11 April 2014.
27. Knublauch, H.; Kontokostas, D. Shapes Constraint Language (SHACL). Available online: https://www.w3.org/TR/shacl/ (accessed on 20 November 2019).
28. Mendel-Gleason, G.; Feeney, K.; Brennan, R. Ontology Consistency and Instance Checking for Real World Linked Data. In Proceedings of the the 2nd Workshop on Linked Data Quality co-located with 12th Extended Semantic Web Conference (ESWC 2015), Portorož, Slovenia, 1 June 2015.
29. Poveda-Villalón, M.; Gómez-Pérez, A.; Suárez-Figueroa, M. OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. *Int. J. Semant. Web Inf. Syst. IJSWIS* **2014**, *10*, 7–34. [CrossRef]
30. Lindsay, J.B. WhiteboxTools User Manual. Available online: https://jblindsay.github.io/wbt_book/preface.html (accessed on 4 July 2020).
31. Lu, Y.; Qin, C.-Z.; Zhu, A.-X.; Qiu, W. Application-matching knowledge based engine for a modelling environment for digital terrain analysis. In Proceedings of the GeoInformatics, Hong Kong, China: The Chinese University of Hong Kong, Hong Kong, China, 15–17 June 2012.
32. Sun, K.; Zhu, Y.; Pan, P.; Hou, Z.; Wang, D.; Li, W.; Song, J. Geospatial data ontology: The semantic foundation of geospatial data integration and sharing. *Big Earth Data* **2019**, *3*, 269–296. [CrossRef]
33. Peng, Y.; Wei, Y.; Di, L.; He, L.; Gong, J.; Zhang, L. Sharing geospatial provenance in a service-oriented environment. *Comput. Environ. Urban Syst.* **2011**, *35*, 333–343.
34. Di, L.; Yue, P.; Ramapriyan, H.K.; King, R.L. Geoscience Data Provenance: An Overview. *IEEE Trans. Geosci. Remote Sens.* **2013**, *51*, 5065–5072. [CrossRef]
35. Servigne, S.; Ubeda, T.; Puricelli, A.; Laurini, R. A Methodology for Spatial Consistency Improvement of Geographic Databases. *GeoInformatica* **2000**, *4*, 7–34. [CrossRef]
36. Liang, P.; Qin, C.-Z.; Zhu, A.-X.; Hou, Z.-W.; Fan, N.-Q.; Wang, Y.-J. A case-based method of selecting covariates for digital soil mapping. *J. Integr. Agric.* **2020**, *19*, 2–11. [CrossRef]

37. Qin, C.; Wu, X.; Jiang, J.; Zhu, A.-X. Case-based knowledge formalization and reasoning method for digital terrain analysis—Application to extracting drainage networks. *Hydrol. Earth Syst. Sci.* **2016**, *20*, 3379–3392. [CrossRef]
38. Frank, A.U. Tiers of ontology and consistency constraints in geographical information systems. *Int. J. Geogr. Inf. Sci.* **2001**, *15*, 667–678. [CrossRef]
39. Smith, M.J.D.; Goodchild, M.F.; Longley, P.A. *Geospatial Analysis—A Comprehensive Guide to Principles, Techniques and Software Tools*, 6th ed.; The Winchelsea Press: Winchelsea, UK, 2018.
40. Hutchinson, M.F. Adding the Z Dimension. In *The Handbook of Geographic Information Science*; Wilson, J.P., Fotheringham, A.S., Eds.; Blackwell Publishing Ltd.: Hoboken, NJ, USA, 2008.
41. Tomaszuk, D. RDF Validation: A Brief Survey. In Proceedings of the International Conference: Beyond Databases, Ustroń, Poland, 30 May–2 June 2017.
42. Wiharja, K.; Pan, J.Z.; Kollingbaum, M.J.; Deng, Y. Schema aware iterative Knowledge Graph completion. *J. Web Semant.* **2020**, *65*, 100616. [CrossRef]
43. Gayo, J.E.L.; Prud'hommeaux, E.; Solbrig, H.R.; Boneva, I. Validating and describing linked data portals using shapes. *arXiv* **2017**, arXiv:1701.08924 [cs.DB].
44. Bosch, T.; Eckert, K. Guidance, please! towards a framework for RDF-based constraint languages. In Proceedings of the 2015 International Conference on Dublin Core and Metadata Applications, São Paulo, Brazil, 1–4 September 2015; pp. 95–111.
45. Knublauch, H. SHACL and OWL Compared. Available online: https://spinrdf.org/shacl-and-owl.html (accessed on 2 November 2019).
46. Albrecht, J. Universal Analytical GIS Operations: A Task-Oriented Systematization of Data Structure-Independent GIS Functionality. In *Geographic Information Research: Transatlantic Perspectives*; Onsrud, H., Craglia, M., Eds.; Taylor & Francis: Abingdon, UK, 1998; pp. 577–591.
47. Miles, A.; Bechhofer, S. Simple Knowledge Organization System. Available online: https://www.w3.org/TR/2009/REC-skos-reference-20090818/ (accessed on 6 May 2021).
48. Brauner, J. *Formalizations for Geooperators-Geoprocessing in Spatial Data Infrastructures*; Technische Universität Dresden: Dresden, Germany, 2015.
49. Battle, R.; Kolas, D. GeoSPARQL: Enabling a Geospatial Semantic Web. *Semant. Web J.* **2012**, *3*, 355–370. [CrossRef]
50. Qin, C.; Zhu, A.X.; Pei, T.; Li, B.; Zhou, C.; Yang, L. An adaptive approach to selecting a flow-partition exponent for a multiple-flow-direction algorithm. *Int. J. Geogr. Inf. Sci.* **2007**, *21*, 443–458. [CrossRef]
51. Rampi, L.P.; Knight, J.F.; Lenhart, C.F. Comparison of Flow Direction Algorithms in the Application of the CTI for Mapping Wetlands in Minnesota. *Wetlands* **2014**, *34*, 513–525. [CrossRef]
52. Wilson, J.P. *Environmental Applications of Digital Terrain Modeling*; Wiley-Blackwell: Oxford, UK, 2018.
53. Wang, Y.-J.; Qin, C.-Z.; Zhu, A.-X. Review on algorithms of dealing with depressions in grid DEM. *Ann. Gis* **2019**, *25*, 83–97. [CrossRef]
54. International Organization for Standardization. *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs)—Part 11: Guidance on Usability. ISO 9241-11: 1998*; International Organization for Standardization: Geneva, Switzerland, 1998.
55. Huang, Z. Usability of tourism websites: A case study of heuristic evaluation. *New Rev. Hypermedia Multimed.* **2020**, *26*, 55–91. [CrossRef]
56. Nielsen, J. *Designing Web Usability: The Practice of Simplicity*; New Riders Publishing: Indianapolis, IN, USA, 2000.
57. Nielsen, J. Usability 101: Introduction to Usability. Available online: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (accessed on 1 April 2021).
58. Conejo, R.; Guzmán, E.; Pérez-de-la-Cruz, J.-L. Knowledge-based validation for hydrological information systems. *Appl. Artif. Intell.* **2007**, *21*, 803–830. [CrossRef]
59. Shu, Y.; Liu, Q.; Taylor, K. Semantic validation of environmental observations data. *Environ. Modell. Softw.* **2016**, *79*, 10–21. [CrossRef]
60. Yu, J.; Taylor, P.; Cox, S.J.D.; Walker, G. Validating observation data in WaterML 2.0. *Comput. Geosci.* **2015**, *82*, 98–110. [CrossRef]
61. Tan, W.-S.; Liu, D.; Bishu, R. Web evaluation: Heuristic evaluation vs. user testing. *Int. J. Ind. Ergon.* **2009**, *39*, 621–627. [CrossRef]
62. Maguire, M.; Isherwood, P. A Comparison of User Testing and Heuristic Evaluation Methods for Identifying Website Usability Problems. In Proceedings of the International Conference of Design, User Experience, and Usability, Cham, Switzerland, 4–8 July 2018; pp. 429–438.