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

Replication of the Question-based Spatial Computing Approach

Experiences and Suggestions for Further Developments

vorgelegt von

MSc Selina Studer
104586, UNIGIS MSc Jahrgang 2016

Betreuerin:
Prof. Dr. Barbara Hofer

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

Zürich, 17.02.2019

Science Pledge

By my signature below, I certify that my thesis is entirely the result of my own work. I have cited all sources I have used in my thesis and I have always indicated their origin.

Zürich, 17th February 2019

_____

Selina Studer

Preface

The content of this work contributes to the conception of future GIS that promote spatial analysis in non-GIS specific disciplines. This and the possibility to develop Python skills were the motivation why I chose this topic.

I would like to take this opportunity to thank Prof. Dr. Barbara Hofer supervisor of this thesis and co-author of the manuscript for the supervision, cooperation and especially the encouragement to publish the manuscript. I gratefully acknowledge the GIS section of the ICRC for make available the UrbanWaterToolbox. My thanks also go to my employer, the Swiss Air Force, which supported me with flexible time management. In particular I would like to thank Yves for his patience, his great support through his scientific expertise and his support with Python.

This thesis consists of two main parts: first, a manuscript-based master thesis and second, a report that documents the technical work in detail. The manuscript was authorized by Selina Studer (first author) and Prof. Dr. Barbara Hofer (second author). The contributions of the second author were inputs at the conceptional phase and a substantial revision on the draft paper.

The manuscript was written to be submitted to the International Journal of Digital Earth (Impact Factor 3.024/ 2017) which among other things promotes the development of methods that turn geo-data from scientific to social, into useful information that can be analyzed, which is the aim of the manuscript.

Table of Contents

Science Pledge ..... **II**

Preface ..... **III**

List of Figures ..... **V**

List of Tables ..... **V**

Manuscript ..... **VI**

ABSTRACT **2**

1 Introduction **2**

2 Core Concepts for Spatial Information and the Language for Spatial Computing **3**

3 Case Study and Evaluation Criteria **4**

3.1 The Case Study 5
3.2 Criteria for Assessing the Simplification of the Spatial Analysis 6

4 Results and Evaluation of the Analysis based on the Language for Spatial Computing **7**

4.1 Conventional Analysis and the Analysis with the Language for Spatial Computing 7
4.2 Assessing the Analysis conducted with the Language for Spatial Computing 9

5 Further Observations on the Language for Spatial Computing **11**

6 Conclusions **12**

References **13**

Report ..... **VIII**

1 Introduction **1**

2 Overview of the Procedure **1**

3 Tools and Environments **1**

3.1 Software 1
3.2 Licencing 2

4 Material **2**

4.1 Data 2
4.2 Conventional Analysis 3
4.3 Core Concepts Library 3

5 Methods **5**

5.1 Core Computations 5
5.2 Preprocessing 6
5.3 Do Not Return self 6
5.4 Save to Temporary Workspace 7
5.5 Unique ID 7
5.6 Iterable Objects 8

5.7 New Implemented Core Computations 8
5.8 Helper-Methods 12
6 Limitations 13
7 Conclusions 13
8 References 15
Appendix 16
A. Conventional Analysis 16
B. Analysis with the Core Concepts 17
C. Core Concepts Library 18

List of Figures

Figure 1 The core concepts mediate between the technological layer and the user (application layer) (adapted from Kuhn and Ballatore (2015)). 4
Figure 2 Input data and the resulting buildings within distance and elevation. 5
Figure 3 Conventional analysis (left) and the analysis with the language for spatial computing (right). 8
Figure 4 Overview of the approach of the master thesis. 1
Figure 5 Illustration of the urban water analysis (ICRC, 2017). 3
Figure 6 Overview of the Python package coreconcepts with its Python files. 4
Figure 7 Core computations of the core concepts (based on Kuhn and Ballatore (2015)). 5

List of Tables

Table 1 Overview of the core concepts (Kuhn and Ballatore 2015). 3
Table 2 Overview of the criteria used for the comparison of the analyses. 6
Table 3 New implemented core computations (notation as in Kuhn and Ballatore (2015)). 7
Table 4 Answering sub-questions with the core concepts (left) and the conventional analysis (right). 9
Table 5 Comparison of the implementations of the buffer methods in the earlier implementation (left) and the implementation within this contribution (right). Before, the input was overwritten by a modified self, in the current implementation the method returns a new object. 7
Table 6 Overview of the new implemented core computations and their purposes. 8

Manuscript

Replication of the Question-based Spatial Computing Approach – Experiences and Suggestions for Further Developments

Selina Studer[a]* and Barbara Hofer[a]

[a] Department of Geoinformatics - ZGIS, University of Salzburg, Salzburg, Austria

*Selina Studer – studer.selina@gmx.ch - Department of Geoinformatics - ZGIS, University of Salzburg, Salzburg 5020, Austria

Replication of the Question-based Spatial Computing Approach – Experiences and Suggestions for Further Developments

ABSTRACT Geographic Information Systems (GIS) have developed into complex toolboxes and require analysts to formulate spatial questions according to the requirements of data formats and tools provided by their GIS-application. The recently proposed language for spatial computing aims to provide a question-based and thus more comprehensible approach for spatial analyses that especially supports scientists and experts from other disciplines to conduct spatial analyses in their fields. In this contribution, we apply the question-based spatial computing approach to a case study in the humanitarian field and compare the resulting script to a script written with a conventional GIS tool. The comparison of the two versions of the analysis is based on six criteria covering qualitative and quantitative aspects of the analysis as well as the implementation concept behind the new language. Our results show, that the new approach requires fewer computational steps than the conventional script. In addition, the declarative approach lets users focus on the content of the spatial question and the query-like character of the language makes it in fact more comprehensible. Besides these benefits of the language for spatial computing, observations on challenges of the further development of the language are shared as an outcome of this study.

Keywords: core concepts; language for spatial computing; question-based analysis; domain-specific language; transdisciplinarity

1 Introduction

Spatial analyses are motivated by finding an answer to a question and thus to better understand problems and to support decision making. That's why spatial analyses play an important role in various disciplines including the humanitarian field. The tools for spatial analyses, Geographic Information Systems (GIS), have been developed into complex toolboxes and require analysts to formulate spatial questions according to the requirements of data formats and tools provided by their GIS-application. Deciding which tools and data to use to answer spatial questions distracts users from the core of the question, requires expert knowledge (Scheider, Ballatore, and Lemmens 2018; Kuhn 2012; Albrecht 1989) and reduces resources for critical spatial thinking (Bearman et al. 2016). This prevents non-GIS specialists from carrying out spatial analyses effectively and consequently reduces the use of spatial analyses for knowledge generation in potential GIS application domains (Vahedi, Kuhn, and Ballatore 2016).

Counteracting this situation, Kuhn and Ballatore (2015) developed a question-based approach for spatial analyses with the *language for spatial computing* (Kuhn and Ballatore 2015). This language simplifies spatial analyses with its new perception of space with the core concept (see Section 2) and its query-like nature. The idea behind the language was illustrated in Vahedi et al. (2016) with the structured query language (SQL) that allows users to ask simple questions on relational databases by using the semantics "SELECT *attributes* FROM *tables* WHERE *condition*". The same requirement exists for spatial questions, which must be answered with simple and normative semantics in a content-oriented way. SQL is used cross-disciplinary and achieved a mature self-image and understanding of its tools (Vahedi, Kuhn, and Ballatore 2016).

The language for spatial computing has been implemented in first case studies. Eventually it should result in a high-level programming language that can be used on existing GIS platforms (Vahedi, Kuhn, and Ballatore 2016). Kuhn and Ballatore (2015) requested further research to clarify whether

the underlying core concepts are sufficient to form the language for spatial computing and to examine how core computations of the language for spatial computing mediate between underlying GIS applications, geodata and core concepts of spatial information. With the resulting findings formal specifications of the language for spatial computing and software integration could be revised (Kuhn and Ballatore 2015).

In this contribution, we apply the language for spatial computing to a real-world example in the humanitarian field and compare the resulting Python script to a script containing the conventional ArcPy-analysis. The approach followed in this contribution compares to work presented in Vahedi et al. (2016) and contributes to the request of Kuhn and Ballatore (2015). We also use the Python-implementation of the language for spatial computing of Vahedi et al. (2016) and implement additional computations for our case study. For determining the criteria for comparison in our research we extend the criteria used in Vahedi et al. (2016) and use the following six criteria for the assessment of the simplification through the language for spatial computing: *question-based, computational steps, comprehensibility, role of base language, role of underlying GIS, role of data property.*

The replication of a case study using the question-based spatial computing approach allows to investigate two questions:

1) Can similar conclusions about the benefits of the language be reached for the present application case in comparison to what has been reported in Vahedi et al. (2016)?
2) Which experiences are made with the use of the language for spatial computing that allow suggestions for its further development?

In the following we review the previous work done for the language for spatial computing (Section 2) and introduce the case study and our procedure (Section 3). In Section 4 we present the results that are further discussed in Section 5 before we conclude with Section 6.

2 Core Concepts for Spatial Information and the Language for Spatial Computing

Janelle and Goodchild (2011) identified the need for a clear and simple conceptual view to understand geoinformation. Kuhn (2012) built on this work that captured spatial phenomena in a few abstract concepts and suggested to perceive space with the seven core concepts of spatial information (Table 1) instead of the perspective of GIS applications and data formats. Unlike the geo-atom of Goodchild et al. (2007) that abstracts spatial phenomena to a single form, the core concepts are a content-based abstraction of spatial information. The abstraction level was chosen as high as possible that one can grasp all the concepts at once whereby they still need to make sense (Kuhn 2012). The concepts granularity and accuracy (nos. 6-7 in Table 1) can be applied as quality concepts to the content concepts location, field, object, network and event (nos. 1-5).

Table 1 Overview of the core concepts (Kuhn and Ballatore 2015).

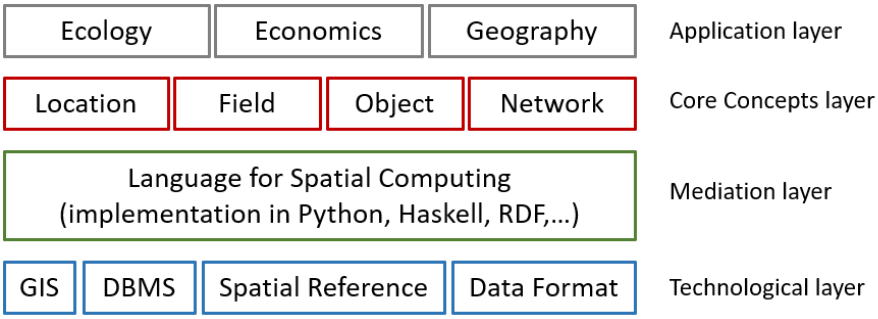
No.	Core Concept	Question	
1	Location	where	*Content concepts*
2	Field	value of a position in space and time	
3	Object	its properties and relations to other objects	
4	Network	connectivity between objects	
5	Event	time or duration in fields, object or network	
6	Granularity	amount of detail in fields, objects, network and events	*Quality concepts*
7	Accuracy	accuracy of information in respect to a reference	

The core concepts of spatial information are the underlying concept of the language for spatial computing (Vahedi, Kuhn, and Ballatore 2016). The language is structured in such a way that geodata is read in as one of the core concepts, depending on the content and the question to be answered. This turns imported geodata into Abstract Data Types (ADTs[1]); the data can be seen as instances of the core concepts available for manipulation with the language for spatial computing. For each core concept a set of *core computations* exists that corresponds to analysis functionality (Vahedi, Kuhn, and Ballatore 2016). The linking of the data with core concepts suggests, which core computations can be applied to the data.

A set of only a few meaningful and combinable core computations reduces the complexity of spatial analyses and allows users to speak a language they are familiar with unlike the technical languages that software often requires (Kuhn and Ballatore 2015). Thereby spatial analyses shift from spatial computing to answering questions, which enhances transdisciplinary use of spatial analyses (Hofer and Scheider under review; Kuhn 2012).

The core computations are a layer implemented on top of existing GIS-applications such as ArcGIS. Thus, the language for spatial computing mediates between the technological layer and the user's perception of spatial information (c.f. Figure 1). The core computations have been implemented in different languages, among others with Python (Kuhn and Ballatore 2015).

Figure 1 The core concepts mediate between the technological layer and the user (application layer) (adapted from Kuhn and Ballatore (2015)).



Vahedi et al. (2016) applied the language for spatial computing to a real example from economy. They used Python to implement the language in form of a Python library named *CoreConcepts*. The library consists of a Python class for each core concept and the core computations were implemented as methods of these classes using the ArcPy library. The authors compared a conventional analysis using ArcPy with an analysis using the language for spatial computing. They highlight the declarative approach of the new language versus the procedural solution of the conventional analysis and also showed a reduction of the computational steps by 45% with the language for spatial computing.

3 Case Study and Evaluation Criteria

This work investigates, whether the findings achieved by Vahedi et al. (2016) - the reduction of

[1] ADT: "class of object whose logical behaviour is defined by a set of values and a set of operations" (Dale & Walker, 1996 in Vahedi et al. 2016).

computational steps and the simplified understanding of spatial analysis with the language for spatial computing - can be confirmed for another case study. This section introduces the real-world case study and the selection of criteria for the assessment of the analysis implemented with the language for spatial computing.

3.1 The Case Study

The case study used in this work is taken from the humanitarian field, where crisis managers often work in interdisciplinary teams and make judgments and decisions under stress (Cai et al. 2006). Therefore, the humanitarian field is a prime example of the need for a simple and established language for spatial computing. The spatial analysis is borrowed from the International Committee of the Red Cross (ICRC) that examines the access of households to water sources and has been used by engineers in the field (ICRC 2017). The purpose of this analysis is to find buildings that are supplied by water points within a distance and elevation parameter. The input data consisted of a Digital Elevation Model (DEM)[2], feature datasets with water points[3], buildings[4], the area of interest and two parameters distance and elevation. Results are a feature layer for each water point containing the buildings within the distance and elevation parameters (visualized in Figure 2). Access to water is an important factor for behavioural responses to hygiene and sanitation measures (Ntozini et al. 2015) and therefore a frequent analysis in the humanitarian field. The case study is hereafter referred to as urban water analysis.

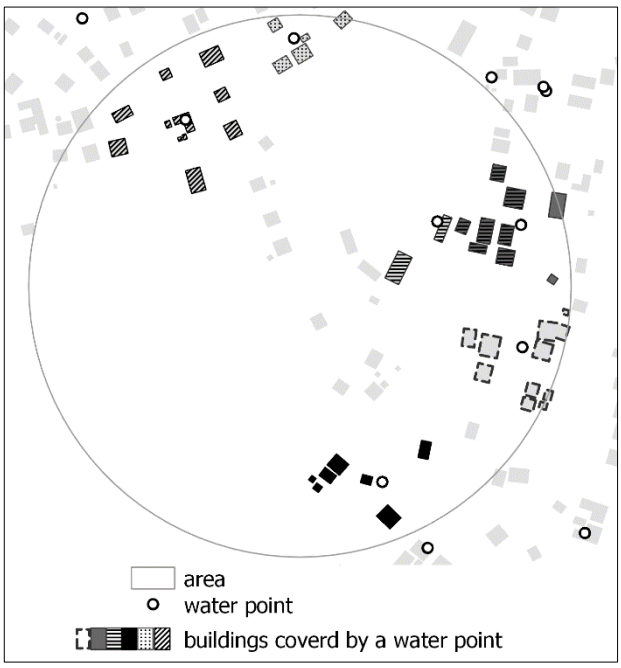


Figure 2 Input data and the resulting buildings within distance and elevation.

[2] SRTM, retrieved from http://dwtkns.com/srtm30m/, accessed 7.12.2018
[3] Akvo National water point mapping Sierra Leone, Retrieved from Humanitarian Data Exchange (HDX): https://data.humdata.org/dataset/national-water-point-mapping-sierra-leone, accessed 7.12.2018
[4] OpenStreetMap Building export, retrieved from HDX: https://data.humdata.org/dataset/hotosm_sierra_leone_buildings, accessed 7.12.2018

The urban water analysis is conducted twice: once with ArcPy (conventional analysis) and once with the language for spatial computing. The Python scripts of the conventional analysis and the core concepts-based implementation can be found in Figure 3. The procedural steps of the conventional urban water analysis consist of the seven steps listed below[5] (c.f. left side of Figure 3).

(1) Select water points within area,
(2) select buildings within area,
(3) extract elevation values of water points to the attribute table of the water points,
(4) calculate centroids of buildings,
(5) extract elevation values of the centroids of the buildings to the attribute table of the buildings,
(6) join the attribute with the elevations to a copy of the building feature class,
(7) select buildings within the distance parameter and elevation parameter with a for-loop through each water point.

For the implementation with the language for spatial computing, the Python implementation was used as this implementation was well documented in previous publications (Kuhn and Ballatore 2015; Vahedi, Kuhn, and Ballatore 2016) and on GitHub[6]. For the back-end implementation of the core computations ArcPy was used. The extensions of the code developed within this research is also available on GitHub[7].

3.2 Criteria for Assessing the Simplification of the Spatial Analysis

To assess the benefits and constraints of the language for spatial computing the question-based analysis was assessed and compared with the conventional analysis. The comparison suggested in Vahedi et al. (2016) considered the criteria *question-based*, *computational steps* and *role of data property*. In this contribution we extend these criteria in order to consider further aspects of the language in the evaluation. The extension of the criteria takes every layer of the conception of the language into consideration (c.f. Figure 1). Table 2 shows the six criteria and how the selected criteria are linked to these four layers.

Table 2 Overview of the criteria used for the comparison of the analyses.

Layer	**Criteria**
Application layer	Question-based Computational steps
Core Concept layer	Comprehensibility
Mediation layer	Role of base language
Technological layer	Role of underlying GIS Role of data property

The criteria *role of underlying GIS* and *role of data property* and *role of base language* examine the effect of the initial technical situation consisting of ArcGIS and Python on the implementation of language for spatial computing. The criterion *computational steps* considers the difference in the

[5] The original analysis from ICRC was adapted and shortened for better clarity.
[6] https://github.com/spatial-ucsb/ConceptsOfSpatialInformation, accessed 13.2.2019
[7] https://github.com/sstuder/QuestionBasedSpatialComputing, accessed 13.2.2019

number of computations between the conventional analysis and the analysis with the core concepts. With the qualitative criteria *question-based* and *comprehensibility* we asses in what sense the language for spatial computing simplifies spatial analyses, for which we assume a user to be a novice with little or no experience in spatial computing. In general, the criteria were evaluated by a detailed assessment and comparison of the operations used in the two analyses and the required adaptations of the core computations.

4 Results and Evaluation of the Analysis based on the Language for Spatial Computing

4.1 Conventional Analysis and the Analysis with the Language for Spatial Computing

Figure 3 shows the conventional ArcPy script of the urban water analysis (left) and the Python script of the analysis based on the core concepts (right). The analyses will be discussed in detail in the next section.

Whereas the available core concepts were sufficient for the selected case study, the core concepts-based implementation required an extension of the existing core computations of the currently available Python library of Vahedi et al. (2016). The four core computations shown in

Table 3 were added to the core concept object:

Table 3 New implemented core computations (notation as in Kuhn and Ballatore (2015)).

CC	Operator: input parameters → output type	Comments
Object	restrictDomain: object x object → object	Restrict an object to the extent of another object
	get: object x (object → value) → value	Get the value of a property of the object
	addProperty: object x field → value	Add the value of a field as an attribute to the object
	withProperty: object x sql → object	Select object with a sql expression

The *get*-method was mentioned in (Kuhn and Ballatore 2015) but not yet implemented. The function *makeObject* existed before but was modified within this contribution to make objects iterable and thus allow parts of an object to be treated as objects, as proposed by Kuhn & Ballatore (2015).

Additionally, we introduced ‘helper-methods’. We implemented the two helper methods *save* and *show*. Helper methods are fundamental methods that are indispensable for coding but are not a core computation. The *save*-method enables intended saving of interim and final results to a file. In earlier ArcPy-based implementations of the language for spatial computing the output of each ArcPy function was saved. In a sequence of questions within a script, permanent storage of interim results is an undesired effect. The *show*-method simplifies the handling by allowing to view an attribute table of a temporary or permanent object quickly in the console and without saving or opening it in a GIS.

Conventional analysis

```
# load input data
area = 'C:/area.shp'
dem = 'C:/dem.tif'
waterPoint = 'C:/waterPoints.shp'
building = 'C:/buildings.shp'

# set parameters
distance = 50
elevation = 3

# select water points within area
waterPoint_inArea = SelectLayerByLocation_management(waterPoint, 'INTERSECT', area)

# select buildings within area
building_inArea = SelectLayerByLocation_management(building, 'INTERSECT', area)

# elevation of waterPoints
waterPoint_elev = ExtractValuesToPoints(waterPoint_inArea, dem, 'in_memory/wp_elev')

# calculate elevation of buildings (using centroid)
building_point = FeatureToPoint_management(building_inArea, 'in_memory/building_point', 'CENTROID')
building_pt_elev = ExtractValuesToPoints(building_point, dem, 'in_memory/building_pt_elev')
building_elevation = CopyFeatures_management(building_inArea, 'in_memory/building_elevation')
JoinField_management(building_elevation, 'FID', building_pt_elev, 'FID', 'RASTERVALU')

cursor = SearchCursor(waterPoint_elev, ['OID@', 'SHAPE@', 'RASTERVALU'])
for wp in cursor:
   geom = wp[1]

   # select buildings within distance
   D = str(distance) + ' Meters'
   inDistance = SelectLayerByLocation_management(building_elevation, 'WITHIN_A_DISTANCE', geom, D)

   # select buildings within elevation
   sql = 'RASTERVALU >= ' + str(wp [2] - elevation) + ' AND ' +
   'RASTERVALU <= ' + str(wp [2] + elevation)
   WDWE = SelectLayerByAttribute_management(inDistance, 'SUBSET_SELECTION', sql)

   #  save output
   CopyFeatures_management(WDWE, 'C:/WDWE_' + str(wp [0]) + '.shp')
```

Core Concepts

```
# load input data
area = makeObject('C:/area.shp')
dem = makeField('C:/dem.tif').restrictDomain(area, 'inside')
waterPoint = makeObject('C:/waterPoints.shp').restrictDomain(area, 'inside')
building = makeObject('C:/buildings.shp').restrictDomain(area, 'inside')

# set parameters
distance = 50
elevation = 3

# Question 1: What are the elevations of the water points?
waterPoint_elev = waterPoint.addProperty(dem)

# Question 2: What are the elevations of the buildings?
building_elev = building.addProperty(dem)

for wp in waterPoint_elev:

   # Question 3: Which buildings are within the distance of the water point?
   wp_buffer = wp.buffer(distance, 'Meters')
   buildings_in_d = building_elev.restrictDomain(wp_buffer, 'inside')

   # Question 4: Which buildings are within the elevation parameter of the  water point?
   wpElev = wp.get('RASTERVALU')
   sql = 'RASTERVALU >= ' + str(wpElev - elevation) + ' AND ' +
   'RASTERVALU <= ' + str(wpElev + elevation)
   WDWE = buildings_in_d.withProperty(sql)

   # save output
   id = wp.get('FID')
   WDWE.save('C:/out', 'WDWE_' + str(id), '.shp')
```

Figure 3 Conventional analysis (left) and the analysis with the language for spatial computing (right).

4.2 Assessing the Analysis conducted with the Language for Spatial Computing

This section provides the evaluation of the analysis conducted with the language for spatial computing based on the criteria introduced in Section 3.2.

4.2.1 Question-based

With regard to the question-based criterion, we can say that the analysis with the core concepts answers specific questions, is declarative instead of procedural and more goal-oriented. This conclusion is based on the following observations.

In case of the analysis with the language for spatial computing the main question was decomposed into four sub-questions and each of it could be answered with one or two computations which implies that with the core computations the specific question can be answered (Table 4). This gives the core computations a more declarative and less procedural character and brings the analysis closer to the objective. The two analyses differ in particular to answer the second question (Table 4). A procedure of four computations was needed in the conventional approach to add the property of height to the buildings (nos. 4-7 in Table 4). With the core computations the user simply can ask for a property of an object with one single goal-oriented computation *addProperty* whereas the procedure to calculate the buildings heights is hidden in the library. *addProperty* directly uses the centroid of a polygons as input and thus prejudges a decision. Whether this is user-friendly can be questioned. But with the procedure in the conventional analysis, the content of the question can easily be lost sight of. This is a prime example which shows that with the core concepts the user can concentrate on answering a question instead of stringing together procedural computations.

The content remote *SearchCursor* function in the conventional analysis that allows the iteration through each feature is also hidden in the library by implementing directly in the *makeObject* function (c.f. Section 4.1). Thus, objects of the core computations are iterable without the user having to perform a computation for which there is no question.

Table 4 Answering sub-questions with the core concepts (left) and the conventional analysis (right).

Sub-question	No.	Core Concepts	No.	Conventional Analysis
			1	`SelectLayerByLocation_management()`
			2	`SelectLayerByLocation_management()`
1) What are the elevations of the water points?	*1*	`addProperty()`	3	`ExtractValuesToPoints()`

2) What are the elevations of the buildings?	*2*	`addProperty()`	*4*
			5
			6
			7
				`FeatureToPoints_management()`
				`ExtractValuesToPoints()`
				`CopyFeatures_management()`
				`JoinField_management()`

Iteration	`for wp in waterPoints`		8	`for wp in SearchCursor()`
3) Which buildings are within the distance of a water point?	*3* *4*	`buffer()` `restrictDomain()`	9	`SelectLayerByLocation_management()`
4) Which buildings are within the elevation parameter?	`sql` *5* *6*	`get()` `withProperty()`	`sql` 10	`SelectLayerByAttribute_management()`
Save output	*7* *8*	`get()` `save()`	11	`CopyFeatures_management()`

4.2.2 Computational Steps

The conventional analysis includes 11 computations whereas the implementation with the core concepts contains 8 computations (7 core computations plus the helper-method *save)* (c.f. Table 4). Thus, the question-based approach is 27% shorter. Although the considerable reduction by 45% in Vahedi et al.

(2016) could not be reached, the reduction of computational steps was verified with this case study as well. Procedural calculations and abstract computations such as the *SearchCursor* function are hidden in the backend of the language for spatial computing and lead to a reduction of computational steps.

The core concepts do not necessarily reduce the amount of computations. Sometimes several core computations need to be combined for achieving a result for which one specific function exists in ArcPy. For example, the ArcPy function *SelectLayerByLocation_management* in the conventional analysis selects features within a distance whereas with the core concepts two core computations *buffer* and *restrictDomain* were combined.

4.2.3 Comprehensibility

By reading in spatial data as a core concept, the number of core computations a user can apply is limited to a certain amount. This makes it easy for a user to grasp, select and apply possible commands to query spatial data. In addition, the syntax of the language for spatial computing is short, concise and the descriptive terms make it intuitive and thus easy for novices to understand and learn the language. According to Ihaka and Gentleman (1996), the syntax of a computer language is only superficial, but determines the way in which users of the language express themselves. Like this, the core concepts are not a newly programmed GIS, but with the language for spatial computing, they offer a new *superficial* approach to how users perceive and query space.

4.2.4 Role of Base Language

Besides the functions and methods implemented within the Python library CoreConcepts, a user could also use any Python syntax such as conditional statement, enumerate items, import other libraries and so on. In case a question cannot be answered with the core computations, other libraries could be used to extend the analysis by more advanced users. The embedding in Python gives many possibilities, but it also requires that all users gain some basic Python-knowledge. At least a user needs to know how to write a value to a variable, how to type strings, use methods, for-loops and know the rules of indentations. Implementations in other languages like Haskell, RDF or JavaScript require the same basic knowledge of the base language.

4.2.5 Role of Underlying GIS

Underlying GIS have a big influence on the implementation of the core concepts, because the existing functions are directly linked to core computations. In our case study the core computations depend on ArcPy peculiarities such as the function *ExtractValuesToPoints* behind the *addProperty*-computation that generates an extra field named “RASTERVALU”, the temporary memory "in_memory” or the SQL dialects used in ArcGIS[8]. Thus, characteristics of the underlying ArcPy library have an impact on the implementation of the core concepts and would be different with another GIS. Additionally, as recognized by Müller (2015), there is the difficulty that GIS operations are not standardized and depending on the GIS on which the core concepts are built, underlying operations do not perform exactly the same.

[8] http://pro.arcgis.com/en/pro-app/help/mapping/navigation/write-a-query-in-the-query-builder.htm, accessed 13.2.2019

4.2.6 Role of Data Property

In the conventional analysis, data layers are loaded into a GIS-application. As many GIS operations are format dependent, data properties like data format or spatial reference lead to unnecessary conversions (Vahedi, Kuhn, and Ballatore 2016). Presumed that the water points in the case study were stored in a text-file and the area of interest was a land use class in a classified raster, in the conventional analysis several conversion steps would be needed before the spatial analysis could be conducted on the data. The text file would be converted to a feature and the raster class to a polygon. With the core concepts, the user views data from the perspective of the core concepts and choses the most appropriate to compute on the data to answer the spatial question. Vahedi et al. (2016) stated that data formats do not limit a core computation to be performed. Therefore, the core concepts must be implemented such that all data formats can be read in for each core concept. For our example this means, that when importing a text file as an object, the columns for the coordinates are automatically requested. Or, if a classified raster is read in as an object, the classes are converted to polygons in the back-end.

5 Further Observations on the Language for Spatial Computing

With regard to the further development of the language for spatial computing, several aspects have to be taken into account, which are discussed in more detail here. According to Kuhn and Ballatore (2015) the core concepts aim to reduce the complexity of spatial computations to a low number by defining a set of core computations for each core concept. These operators form the semantic primitives of the language for spatial computing, which can be combined to conduct more complex spatial analyses. The authors provided a set of 29 core computations (Kuhn and Ballatore 2015). Vahedi et al. (2016) introduced three computations and we also added three core computations to conduct the specific case studies. To date, it has never been precisely described according to which criteria the number of core computations should be defined. If the core computations can be arbitrarily extended, they end up end in a confusing number and the goal of a simple approach to spatial analysis was missed. The difficulty now is to provide a number of relevant and universally applicable core computations, which offer the user a maximum of analysis possibilities when combined. Once this has been achieved, it is worth defining the core computations normatively and thus making the geoprocessing functionalities semantically interoperable. This would be the first top-down approach in the field of geoinformation as stated in Kuhn and Ballatore (2015).

A comprehensive language for spatial computing must take interoperability concerning the syntactic, meta and semantic level into account. This requires that the language for spatial computing can be used on different platforms, that a documentation of what a geoprocessing operation does exists, and that designations of an operation across platforms guarantee that exactly the same process is performed. Due to the lack of meaningful descriptions what geoprocessing tools do with the data, users are often forced to acquire knowledge through the backend systems. Consistent behaviour of spatial computations across all platforms is desirable from a user's perspective and could be guaranteed with a common standard (Müller 2015). The implementation of a language for spatial computing is an ideal possibility to start with a standard for geoprocessing functionalities.

The implementation of the core concepts based on existing GIS applications will pose some challenges with regard to the semantic interoperability of these implementations. In the case study, ArcPy peculiarities like the "in_memory" workspace or the automatic generation of an attribute name were integrated in the implementation of the language for spatial computing. Implementations of normatively defined functions based on existing GIS applications are challenging and we question whether

it makes sense to implement the language for spatial computing based on existing GIS applications, as proposed in previous work and if it would not be better to build a new language from scratch.

The language for spatial computing is supposed to work with any data format no matter if the data is provided file-based or as linked data. Additional implementations that test a complete list of different data formats are needed. These tests needed to include how the core concepts deal with raster read in as objects or vectors as fields, networks or as another core concept.

Within this contribution, helper methods were introduced. They do not perform spatial computations neither do they belong to a core concept. In our opinion, helper methods such as *save*, *showAttributes*, *plotMap* or *delete* are absolutely necessary as a user needs computations apart from the core computations for handling the data and memory usage. A final list of helper methods must be drawn up.

6 Conclusions

Our replication of the approach of Vahedi et al. (2016) using a case study from the humanitarian field showed that the language for spatial computing reduces the number of computing steps to conduct the analysis for the case study. The evaluation indicates that the use of core concepts encourages interdisciplinary use of spatial analysis among non-GIS specialists, because of its comprehensibility and its question orientation. Based on the experience made with the case study, we would expect that it is easier for the GIS section of the ICRC to describe to their engineers how data can be queried with the language for spatial computing than developing and distributing ArcGIS script-tools.

The implementation of additional core computations that were required for the case study, led to suggestions for the future implementation of the language for spatial computing. These suggestions include the specification of the required number of core computations, the addition of helper methods, and normative definitions of geoprocessing functionalities. The strong dependency of the core computations on the underlying GIS poses a challenge in case a standardised language is targeted.

Making a universal language for spatial computing accessible to diverse user communities holds a great potential that spatial analyses can increasingly be taken into account in decisions, no matter in which discipline.

References

Albrecht, Jochen
1989 Universal Analytical GIS Operations — a Task-Oriented Systematization of Data Structure-Independent GIS Functionality. *In* Onsrud H., Craglia M. (Eds) Geographic Information Research: Transatlantic Perspectives. Pp. 577–591. Tyalor and Francis.

Bearman, Nick, Nick Jones, Isabel André, Herculano Alberto Cachinho, and Michael DeMers
2016 The Future Role of GIS Education in Creating Critical Spatial Thinkers. Journal of Geography in Higher Education 40(3): 394–408.

Cai, Guoray, Rajeev Sharma, Alan M. MacEachren, and Isaac Brewer
2006 Human-GIS Interaction Issues in Crisis Response. International Journal of Risk Assessment and Management 6(4/5/6): 388.

Dale, Nell, and Henry M. Walker
1996 Abstract Data Types: Specifications, Implementations, and Applications. Jones & Bartlett Learning.

Goodchild, Michael F., May Yuan, and Thomas J. Cova
2007 Towards a General Theory of Geographic Representation in GIS. International Journal of Geographical Information Science 21(3): 239–260.

Hofer, Barbara, and Simon Scheider
under review Geospatial Information Online Processing. *In* Manual of Digital Earth.

ICRC
2017 Calculating Buildings Being Supplied by a Water Point. Draft tutorial about Urban Water Toolbox, Geneva, Switzerland.

Janelle, Donald G., and Michael F. Goodchild
2011 Concepts, Principles, Tools, and Challenges in Spatially Integrated Social Science. The SAGE Handbook of GIS and Society. Thousand Oaks, CA: SAGE: 27–45.

Kuhn, Werner
2012 Core Concepts of Spatial Information for Transdisciplinary Research. International Journal of Geographical Information Science 26(12): 2267–2276.

Kuhn, Werner, and Andrea Ballatore
2015 Designing a Language for Spatial Computing. *In* AGILE 2015. Fernando Bacao, Maribel Yasmina Santos, and Marco Painho, eds. Pp. 309–326. Lecture Notes in Geoinformation and Cartography. Springer International Publishing. http://link.springer.com/chapter/10.1007/978-3-319-16787-9_18.

Kuhn, Werner, Andrea Ballatore, Eric Ahlgren, et al.
2018[2014] Specifications and Resources towards a Language for Spatial Computing: Spatial-Ucsb/ConceptsOfSpatialInformation. Haskell, JavaScript, Python, RDF. spatial@ucsb. https://github.com/spatial-ucsb/ConceptsOfSpatialInformation, accessed September 26, 2018.

Müller, Matthias
2015 Hierarchical Profiling of Geoprocessing Services. Computers & Geosciences 82: 68–77.

Ntozini, Robert, Sara J. Marks, Goldberg Mangwadu, et al.

2015 Using Geographic Information Systems and Spatial Analysis Methods to Assess Household Water Access and Sanitation Coverage in the SHINE Trial. Clinical Infectious Diseases 61(suppl_7): S716–S725.

Scheider, Simon, Andrea Ballatore, and Rob Lemmens
2018 Finding and Sharing GIS Methods Based on the Questions They Answer. International Journal of Digital Earth 0(0): 1–20.

Vahedi, Behzad, Werner Kuhn, and Andrea Ballatore
2016 Question-Based Spatial Computing—A Case Study. *In* Geospatial Data in a Changing World Pp. 37–50. https://link.springer.com/chapter/10.1007/978-3-319-33783-8_3, accessed February 23, 2017.

Report

1 Introduction

A detailed report on the technical work is presented in this part of the thesis. This report includes a general overview of the procedure (Section 2), detailed information on the software (Section 3.1), data (Section 4.1) and scripts (Section 4.2) used. Comments and illustration of adjustments in the implementation of the core concepts can be found in Section 5 .

2 Overview of the Procedure

After an introductory literature research (Step 1 in Figure 4) and familiarization with the language for spatial computing (Step 2), a suitable spatial analysis was sought and a data set compiled for it (step 3). Subsequently, the spatial analysis was processed in a conventional way (Python script using ArcPy) (Step 4) as well as with the core concepts (Step 5). During the research work, criteria were developed for the comparison of the two analyses (Step 6). Finally, the two analyses were compared against the criteria (Step 7). The analysis based on the core concepts was discussed with a view to simplifying spatial analyses.

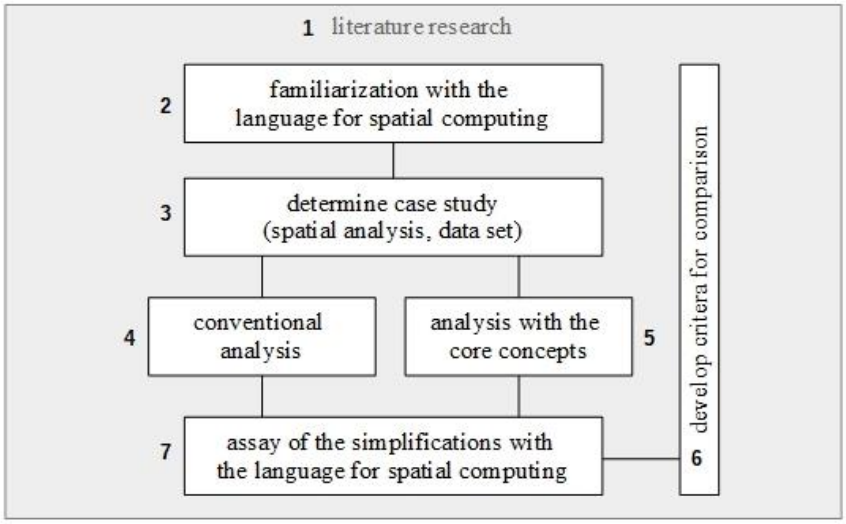


Figure 4 Overview of the approach of the master thesis.

3 Tools and Environments

3.1 Software

The spatial analyses were conducted with ArcGIS Pro 2.2.4 and its Python 3.6.5 integration. The spatial analyses were conducted with the ArcPy library. Git 2.18.0.windows.1 was used for

version control in order to reproduce changes made in the code. PyCharm 2018.2.3 connected to Git served as IDE. Further a 64 bit computer with the operating system Windows 10 Pro was used.

3.2 Licencing

The previous implementations of the core concepts were published under the Apache License 2.0 license by Kuhn, Ballatore, Ahlgren, Thiemann, Zimmer, Vahedi, Hervey, Lafia and Jiang (2018). Code published within this thesis is subject to the license Apache Licence 2.0[9].

4 Material

4.1 Data

All data used for the case study are free data. The Digital Elevation Model (DEM) was used from the NASA Shuttle Radar Topography Mission (SRTM) with a resolution of 1-arcsecond (approximately 30m), EPSG 4326, GeoTIFF-format (Jarvis et al. 2008). The Tile N08W12 was downloaded from NASA servers via the tile downloader of Derek Watkins[10]. This DEM was ok for this case study, which focused on technical aspects. For an urban water analysis with strong interest in the height difference between water points and buildings, a DEM with a higher resolution is recommended.

The water points[11] and the buildings[12] were downloaded as shapefiles with the spatial reference EPSG 4326 from the Humanitarian Data Exchange platform (HDX) that is provided by the Centre for Humanitarian Data. The water point dataset contains a national water point mapping conducted 2012 in Sierra Leone and is provided by akvo.org. The buildings are an OpenStreetMap export of Sierra Leone that fulfil the query "building IS NOT NULL".

[9] http://www.apache.org/licenses/LICENSE-2.0, accessed 28.12.2018
[10] SRTM tile downloader: http://dwtkns.com/srtm30m/, accessed 7.12.2018
[11] Akvo National water point mapping Sierra Leone, Retrieved from HDX: https://data.humdata.org/dataset/national-water-point-mapping-sierra-leone, accessed 7.12.2018
[12] OpenStreetMap Building export, retrieved from HDX: https://data.humdata.org/dataset/hotosm_sierra_leone_buildings, accessed 7.12.2018

4.2 Conventional Analysis

The original of the conventional analysis, the urban water analysis, was provided by the GIS section of the International Committee of the Red Cross based in Geneva, Switzerland. They designed an ArcGIS script-tool that calculates the number of buildings being or not being supplied by a water point by considering the two parameters maximum elevation and maximum distance difference in meters from a water point (see Figure 5) (ICRC, 2017). The output of this script-tool is a report for each water point covering a number of buildings in form of a text file.

For simplification, the original urban water analysis has been shortened so that the output contains only buildings within the elevation and distance parameter and no buildings that fulfil only one or none of the parameters. The output format was also adapted from a text file to a shape file. The analysis used is given in Annex A.

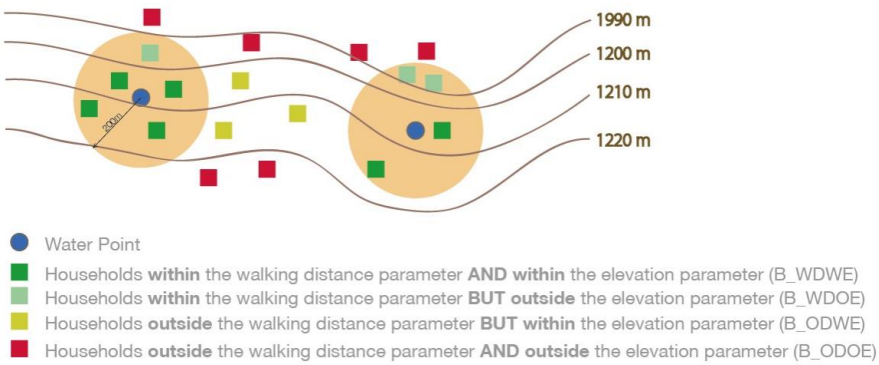


Figure 5 Illustration of the urban water analysis (ICRC, 2017).

4.3 Core Concepts Library

The implementation of the core concepts within this research is based on previous implementations that were published on GitHub. The Python implementation using the ArcPy library was developed within this case study. Existing code was forked and downloaded from the GitHub repository from the Center for Spatial Studies at the University of California, Santa Barbara[13]. The code modified within this thesis was then published again on GitHub.[14]

[13] https://github.com/spatial-ucsb/ConceptsOfSpatialInformation/tree/master/CoreConceptsPy/ArcPy, accessed 19.12.2018

[14] https://github.com/sstuder/QuestionBasedSpatialComputing, accessed 13.02.2019

The structure of the Python package coreconcepts consists of four Python files: utils.py, coreconcepts.py, fields.py and objects.py (compare Figure 6). The files for the implementation of the remaining content concepts (network.py, event.py and location.py) are not yet implemented.

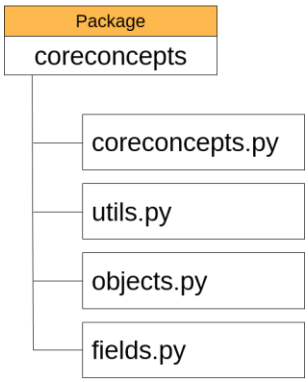


Figure 6 Overview of the Python package coreconcepts with its Python files.

The file utils.py is the file to be imported in a Python script (`from coreconcepts.utils import *`). It is the initial file that defines the key functions with which the user can read in data as a core concept: `makeField()` and `makeObject()`. Based on the ending of the filepath of the data a Python class is returned that is defined in fields.py or objects.py. This returned classes are the Abstract Data Types (ADTs), the instances of the core concepts that can be manipulated with the language for spatial computing.

For each class the Python methods are defined that can be applied to the data. In the coreconcepts.py file the abstract classes for the core concepts are defined. Within the core concept class the class properties *filepath*, *sObj* and *domain* are defined. For the core concept *object* this looks for example like this:

```
class CcObject(object):
    """
    Abstract class for core concept 'object'
    """
    def __init__(self, filepath, objIndex, domain):
        """
        :param filepath: data file path
        :param objIndex: unique ID
        :param domain: desc.extent of the geo_object
        """
        self.filepath = filepath
        self.sObj = objIndex
        self.domain = domain
```

The filepath contains the filepath to the data. sObj is a unique id for each object and domain contains the spatial extent of the data. Each time, a method is applied to data, these properties are updated for the processed data.

Further details and script excerpts that contain the modifications of this contribution are presented in Section 5 The Python package, which was developed in the context of this contribution, can be found in Appendix C or as mentioned before on GitHub.

5 Methods

5.1 Core Computations

Up to now the 35 core computations (c.f. Figure 7) are implemented. Whereas three core computations were newly implemented within the case study of Vahedi et al. (2016) and another three core computations were newly implemented within this case study. Major changes made to the coreconcepts package are explained in more detail in the following sections.

Core Concept	**Operator: input parameters → output type**	**Comments**
Location	istAt: figure x ground → Bool	The contact relation
	isIn: figure x ground → Bool	The containment relation
	Position: figure → point	A point positioning the figure
	Bounds: figure → shape	A shape bounding the figure
Field	new: [(pos,val)] × ([(pos × val)] × pos → val) → field	Interpolating a field from a list of values
	bounds: field → shape	The domain of the field
	getValue: field × pos → val	The value at a position
	local: field × (val → val') → field	Computing new values at all positions
	focal: field × (pos → val') → field	Computing new values from neighbourhoods
	zonal: field × (pos → val') → field	Computing new values from zones
	restrictDomain: field x object → field	Restrict a field to the extent of a object
Object	get: object × (object → val) → value	Get the value of a property of the object
	is: object × object × (object × object → Bool) → Bool	Are the two objects in the given relation?
	same: object × object → Bool	Are the two objects the same?
	buffer: object x val → object	Creates buffer polygons around an object with a distance
	restrictDomain: object x object → object	Restrict an object to the extent of another object
	addProperty: object x field → val	Add the value of a field as an attribute to the object
	withProperty: object x sql → object	Select object according to a sql query to its attributes
Network	nodes: network → [node]	All nodes in a network
	edges: network → [edge]	All edges in a network
	addNode: network × node → network	Add a node to a network
	addEdge: network × edge → network	Add an edge to a network
	degree: network → node → Int	Degree of a node in the network
	connected: network × node × node → Bool	Are two nodes connected?
	shortestpath: network × node × node → [node]	The shortest path between two nodes
	distance: network × node × node → Int	The network distance (as number of nodes)
	breadthfirst: network × node × Int → [node]	All nodes at distance from a node
Event	when: event → date	The time of an event as a date
	within: event → period	The time of an event as a period
	same: event × event → Bool	Are the two events the same?
	during: event × event → Bool	Is the first event happening during the second?
	before: event × event → Bool	Is the first event happening before the second?
	after: event × event → Bool	Is the first event happening after the second?
	overlap: event × event → Bool	Does the first event overlap the second?
Granularity	coarsen: field x (val, val) → field	Coarsening the granularity of a field

Legend:

Implemented in our case study

Implemented in Vahedi et al. (2016)

Figure 7 Core computations of the core concepts (based on Kuhn and Ballatore (2015)).

5.2 Preprocessing

The Python implementation of the core concepts taken from GitHub were submitted in a PyCharm project and packaged up. The previous implementation used ArcMap's Python 2. Thus first, the code was adapted for Python 3 used in ArcGIS Pro. This concerned the *super* functions in the implementation of objects and fields (objects.py, fields.py):

Python 2	**Python 3**
`super(ArcShpObject, self).__init__(filepath, objIndex, domain)`	`super().__init__(filepath, objIndex, domain)`
`super(GeoTiffField, self).__init__(filepath, geoObject)`	`super().__init__(filepath, geoObject)`

I also decided not to use the ArcPy workspace (*arcpy.env.workspace*) as the definition of the workspace was a hard coded value. Therefore, in this research the property filepath of the data was used, in contrast to earlier implementations where the property filename was used.

5.3 Do Not Return self

The previous implementation of a method modifies the input and overwrites the original input by returning *self*. For example, the method buffer, that is coded in objects.py buffers the object and returns *self*. Thus, calling the method for an object called area and run it with the parameters (20, "Meters"), overwrites the original object area with the buffered *self* (compare Table 5, left). Thus, the original object area cannot be further used, for example to be buffered a second time, because it simply does not exist anymore. It would have to be read in a second time as an object.

```
area = make_object("area.shp")
area_buffer_20m = area.buffer(20, 'Meters')
area_buffer_50m = area.buffer(50, 'Meters')
```

Therefore, all methods of the classes fields and object were rewritten (Table 5, right). The object or fields on which the methods were applied, return a temporary file. The filename is composed of a string unique to the method (buf_), an index unique to the object/field (str(self.sObj)) and of the input variable (str(distance)). Like this it can be guaranteed, that the filepath saved to a temporary workspace cannot be overwritten.

Table 5 Comparison of the implementations of the buffer methods in the earlier implementation (left) and the implementation within this contribution (right). Before, the input was overwritten by a modified self, in the current implementation the method returns a new object.

before	after

`def buffer (self, distance, unitType):`
`  """`
`  Buffer input object`
`  @param distance a distance extent to buffer`
`  @param unitType unit type`
`  """`
 	
`  outcome = "\_buffer_"`
`  # determine save file path`
`  outputLocation = self.filepath + outcome +`
`  self.filename`
 	
`  # calculate buffer`
`  concatDistance = str(distance) + " "`
`  + unitType`
`  arcpy.Buffer_analysis(self.filename,`
`  outputLocation, concatDistance)`
 	
`  # update cc instance's attributes`
`  desc = arcpy.Describe(outputLocation)`
`  self.domain = desc.extent`
`  self.filepath = outputLocation`
`  self.filename = os.path.basename(`
`  outputLocation)`
 	
`  return self`
	`def buffer (self, distance, unitType ):`
	`  """`
	`  Buffer input object`
	`  @param distance: buffer distance`
	`  @param unitType: unit type`
	`  """`
	
	`  # determine temporary unique file`
	`  distName = str(distance)`
	`  distName2 = distName.replace(".", "_")`
	`  print("distName2", distName2)`
	`  name = "buf_" + str(self.sObj) + distName2`
	`  outputLocation = "in_memory\\" + name`
	
	`  # calculate buffer`
	`  concatDistance = str(distance) + " "`
	`  + unitType`
	`  Buffer_analysis(self.filepath,`
	`  outputLocation, concatDistance)`
	`  bufObj = utils.makeObject(outputLocation)`
	
	`  # update cc instance's attributes`
	`  desc = Describe(outputLocation)`
	`  bufObj.domain = desc.extent`
	`  bufObj.filepath = outputLocation`
	`  bufObj.filename = os.path.basename(`
	`  outputLocation)`
	
	`  return bufObj`

5.4 Save to Temporary Workspace

Using ArcPy the outputs of most functions are saved to a file. In a Python script it is often not desired, that interim results are saved to a file as the interest is in the final output. The same applied to the conventional analysis and interim results were saved to the ArcPy specific in_memory-workspace[15]. The in_memory-workspace was also applied to the core concepts implementation. So that interim results no longer had to be saved locally. This was solved by saving the output of a method to the in_memory-workspace, like for example of the method buffer: `outputLocation = "in_memory\\buf_14047239163001650"`. Concerning space complexity, this may not be suitable for processing larger data sets.

5.5 Unique ID

To prevent data from being overwritten unintentionally, a unique id is assigned to each data set read in as core concept. This unique id is assigned with the Python built-in function *id()* while

[15] https://pro.arcgis.com/en/pro-app/help/analysis/geoprocessing/modelbuilder/the-in-memory-workspace.htm, accessed 19.12.2018

reading in the data with the function *makeField* or *makeObject*. This unique id is assigned to the class as a property named sObj. This property sObj then can be used to give unique names to the outputs of methods.

5.6 Iterable Objects

It was already mentioned in Kuhn and Ballatore (2015) that parts of objects can be treated as objects itself. Thus, the function *makeObject* in objects.py was modified that objects become iterable and if iterated, each feature becomes an object itself. This was implemented by applying the SearchCursor function on the object and iterate through each OID:

```
class ArcShpObject(CcObject):
    """
    Concrete class for core concept 'object'
    For handling .shp files and feature classes of a geodatabase
    """

    def __init__(self, filepath, objIndex, domain):
        super().__init__(filepath, objIndex, domain)
        self.filepath = filepath
        self.sObj = objIndex
        self.domain = domain
        self.filename = os.path.basename(filepath)
        self.OIDs = SearchCursor(self.filepath, "OID@")

    def __iter__(self):
        return self

    def __next__(self):
        try:
            next_FID = next(self.OIDs)[0]
        except StopIteration:
            self.OIDs = SearchCursor(self.filepath, "OID@")
            raise StopIteration
        next_filepath = f"{self.filepath}_FID={next_FID}"
        MakeFeatureLayer_management(self.filepath, next_filepath)
        SelectLayerByAttribute_management(next_filepath, "NEW_SELECTION", f"FID={next_FID}")
        return ArcShpObject(next_filepath, id(next_filepath), self.domain)
```

5.7 New Implemented Core Computations

Four new core computations were implemented for the spatial analysis of this case study, all of them for the core concept object. These four core computations *restrictDomain*, *get*, *addProperty* and *withProperty* were implemented as Python methods of the class ArcShpObject within the Python file objects.py.

Table 6 Overview of the new implemented core computations and their purposes.

CC	Operator: input parameters → output type	Comments
Object	restrictDomain: object x object → object	Restrict an object to the extent of another object
	get: object x (object → value) → value	Get the value of a property of the object

	addProperty: object x field → value	Add the value of a field as an attribute to the object
	withProperty: object x sql → object	Select object with a sql expression

Core Computation: restrictDomain

The core computation *restrictDomain* already existed for the core concept field but not for the core concept object. For the case study the objects water points and buildings were restricted to the area of interest with the core computation *restrictDomain*. Input of the *restrictDomain* method is the object to be restricted (self), an object to which's extent the self-object is restricted (area-object) and an operation that can be "inside" or "outside". In the following the definition of the *restrictDomain* method can be found. The method uses the ArcPy function *SelectLayerByLocation_management*. Output are all objects within the intersection of the self-object and the area-object. If the operation "outside" is chosen, the selection is switched in an additional step.

```
def restrictDomain(self, object, operation):
  """
  Restricts current instance's domain based on object's domain
  @param object: extent to which the object is restricted
  @param operation: valid options: "inside", "outside"
  """

  name = "restDom_" + str(self.sObj)
  outputLocation = "in_memory\\" + name

  if operation == 'inside':
    # select by location
    select = SelectLayerByLocation_management(self.filepath, "INTERSECT", object.filepath)
    CopyFeatures_management(select, outputLocation)
    restDom = utils.makeObject(outputLocation)

  elif operation == 'outside':
    # select by location
    sel = SelectLayerByLocation_management(self.filepath, "INTERSECT", object.filepath)
    select = SelectLayerByLocation_management(sel, "INTERSECT", object.filepath, "",
    "SWITCH_SELECTION")
    CopyFeatures_management(select, outputLocation)
    restDom = utils.makeObject(outputLocation)

  else:
    raise NotImplementedError(operation)

  # update cc instance's attributes
  desc = Describe(outputLocation)
  restDom.domain = desc.extent
  restDom.filepath = outputLocation
  restDom.filename = os.path.basename(outputLocation)

  return restDom
```

#TODO:

- Implement for other data formats read in as objects, like for example CSV, GeoTIFF or GeoJSON.

Core Computation: get

The core computation *get* gets a value of a property (attribute column) of the object. This method only can be used if an object consists of one feature or if a for-loop is used that iterates through each feature as the method only returns one value of a specific column of the object. With the *SearchCursor* function the specific column is located and the value retuned. For this method, the value needs to be part of the attribute table of the object. This requires that a user knows attribute tables, knows how to write values to it and knows how to examine it. To write values to the attribute table is described in the next core computation (*addProperty*). An attribute table can be examined with the helper method *show* (Section 5.8).

In the case study the *get* method was used in the for-loop through each water point. The height of each water point was queried in order to use it in the sql statement where the value is compared with the elevation of the buildings.

```
def get(self, prop):
  """
  :param: name of the property
  :returns: value of property in the object
  """

  with SearchCursor(self.filepath, prop) as cursor:
    for row in cursor:
      return row[0]
```

#TODO:

- Implement for other data formats read in as objects, like for example CSV, GeoTIFF or GeoJSON.

Core Computation: addProperty

The core computation *addProperty* adds a value to the attribute table of an object. Behind this method the ArcPy function *ExtractValuesToPoint* is used. This function automatically adds a field to the attribute table of the object with the name “RASTERVALU” which is a peculiarity of ArcPy. If the shape type of the object is “point”, the value at the location of the point is added to the attribute table. If the shape type is “polygon”, the polygons first are converted to points, located at the centroid of the polygon. Then the “RASTERVALU” and the values are added to these points before the field “RASTERVALU” again is joined to the polygons. This is an illustrative example where procedural computations are hidden from the user in the library of the coreconcepts.

The *addProperty* method is used in the case study to add the height value from the DEM to the water points and the building polygons.

```
def addProperty(self, in_raster):
  """
  get value of a field and write it to a column named RASTERVALU in the object
  @param in_raster: raster where the value is taken from
  """

  desc = Describe(self.filepath)
  name = "addProperty" + str(self.sObj)
  outputLocation = "in_memory\\" + name

  if desc.shapeType == "Point":
    ExtractValuesToPoints(self.filepath, in_raster.filepath, outputLocation)
    addProperty = utils.makeObject(outputLocation)

  elif desc.shapeType == "Line":
    raise NotImplementedError(desc.shapeType)

  elif desc.shapeType == "Polygon":
    polyToPoint = "in_memory\\polyToPoint_" + str(self.sObj)
    FeatureToPoint_management(self.filepath, polyToPoint, "CENTROID")
    valueToPoint = "in_memory\\valueToPoint_" + str(self.sObj)
    ExtractValuesToPoints(polyToPoint, in_raster.filepath, valueToPoint)
    CopyFeatures_management(self.filepath, outputLocation)
    JoinField_management(outputLocation, "FID", valueToPoint, "FID", "RASTERVALU")
    addProperty = utils.makeObject(outputLocation)
    Delete_management(polyToPoint)
    Delete_management(valueToPoint)

  else:
    raise NotImplementedError("unknown shapeType:", desc.shapeType)

  # update cc instance's attributes
  desc = Describe(outputLocation)
  addProperty.domain = desc.extent
  addProperty.filepath = outputLocation
  addProperty.filename = os.path.basename(outputLocation)

  return addProperty
```

#TODO:

- implement operation “CENTROID” and “INSIDE” for polygons
- implement shapeType == “Line”
- determine own field name instead of “RASTERVALU”
- implement possibility, that several fields can be added to the attribute table
- Implement for other data formats read in as objects, like for example CSV, GeoTIFF or GeoJSON.

Core Computation: withProperty

The core computation *withProperty* extracts all subobjects that fulfil a sql expression. In an implementation, care must be taken to ensure that the used SQL syntax considers the SQL dialects of each possible database management system.

In the case study the *withProperty* method was used to extract the buildings within the elevation parameter.

```
def withProperty(self, sql):
  """
  :param sql: sql expression
  :returns: feature that meets the properties of the sql expression
  """

  name = "wProp_" + str(self.sObj)
  outputLocation = "in_memory\\" + name

  selByAtt = SelectLayerByAttribute_management(self.filepath, "NEW_SELECTION", sql)
  CopyFeatures_management(selByAtt, outputLocation)
  wProp = utils.makeObject(outputLocation)

  # update cc instance's attributes
  desc = Describe(outputLocation)
  wProp.domain = desc.extent
  wProp.filepath = outputLocation
  wProp.filename = os.path.basename(outputLocation)

  return wProp
```

#TODO:

- take into consideration different SQL dialects
- Implement for other data formats read in as objects, like for example CSV, GeoTIFF or GeoJSON.

5.8 Helper-Methods

While implementing the spatial analysis with the core concepts, the need for helper-methods arose. By saving interim results to a temporary workspace (Section 5.4) a method to save the final result to a file in a specific format became compelling. But save is not a core computation, that belongs to one core concept but can be used for all the results. That's why we introduced helper-methods that are fundamental methods that are indispensable for coding but are not an element of the core computations. For the sake of simplicity the method save was implemented as method of a core concept. But it may also be considered to implement helper methods as helper functions, Python functions that can be applied on all the core concepts.

A second helper-method *show* was also implemented. This method allows to examine the attributes of a (temporary) object in the console without the need to save the data to a file and open it in a GIS to view and examine it.

helper method: **save**

```
def save (self, Output_Folder, Output_Name, extension):
  outputLocation = Output_Folder + "\\" + Output_Name + extension
  CopyFeatures_management(self.filepath, outputLocation)
```

helper method: **show**

```
def show(self):
  print("\n")
  print("show 5 first table rows for file:", '\x1b[1;36m' + self.filepath + '\x1b[0m')

  list = []
  fields = ListFields(self.filepath)
  for field in fields:
    list.append(field.name)

  list.remove("Shape")
  header = []
  for field in list:
    header.append(str('{:_^20}'.format(field)))
  print(header)

  count = 1
  with SearchCursor(self.filepath, list) as cursor:

    line = []
    for row in cursor:
      for col in row:
        line.append(str('{:^20}'.format(col)))
      print(line)
      line = []
      if count >= 5:
        break
      count += 1

  del cursor
```

#TODO:

- Implement other helper-methods like *plotMap*
- implement for other data formats read in as object: CSV, GeoTIFF, GeoJSON…
- implement for other core concepts: location, field, network, event

6 Limitations

The language for spatial computing was also implemented in JavaScript, Haskell and RDF. A comparison of the implementations of the language for spatial computing with the different base languages would be important, so that the language does not drift apart into different variations already at the beginning.

In addition, a developer's view of implementing the language for spatial computing could improve performance, feasibility, and time and space complexity. The considerations made in this thesis will serve as support for future implementations of the language for spatial computing.

7 Conclusions

In summary, I would like to reiterate the most important findings of the technical report of this research which can be used as recommendations for future developments.

- It is important, that a language for spatial computing does not return *self* in its computations but writes the result to a new object, field, network… so that the original data can continue to be used.
- Spatial analysis with the language for spatial computing often relies on the combination of core computations. Interim results are not of interest and should be saved only to a temporary memory. It must be borne in mind that large data sets fill the working memory.
- Apart of the core computations further computations like helper-methods are needed. Helper methods do not belong to any core concept. They allow operations like intended saving or check of interim results.
- Within this contribution the function *makeObject* was expanded by the ability to make each object iterable through its features. This was considered in earlier researches of Kuhn and Ballatore (2015).
- Four new core computations *restrictDomain*, *get*, *addProperty* and *withProperty* were implemented within this contribution. How the core computations are to be composed is a component of future research (compare Section 5 in Manuscript).
- All core computations need to be implemented for all possible formats. This means for example also that the core computation *get* needs an implementation for a GeoTIFF that was read in as an object. Either such a method can be realized or a helpful error message provides a solution to the user.
- Investigations for the optimal implementation of the language for spatial computing under consideration of the time and space complexity must be carried out.

8 References

ICRC
2017 Calculating Buildings Being Supplied by a Water Point. Draft tutorial about Urban Water Toolbox, Geneva, Switzerland.

Jarvis, Andy, Hannes Isaak Reuter, Andy Nelson, and Edward Guevara
2008 Hole-Filled Seamless SRTM Data V4. International Centre for Tropical Agriculture (CIAT). http://srtm.csi.cgiar.org.

Kuhn, Werner, and Andrea Ballatore
2015 Designing a Language for Spatial Computing. *In* AGILE 2015. Fernando Bacao, Maribel Yasmina Santos, and Marco Painho, eds. Pp. 309–326. Lecture Notes in Geoinformation and Cartography. Springer International Publishing. http://link.springer.com/chapter/10.1007/978-3-319-16787-9_18.

Kuhn, Werner, Andrea Ballatore, Eric Ahlgren, MarcThiemann, Michel Zimmer, Behazd Vahedi, Thomas Hervey, Sara Lafia, Liangcun Jiang
2018[2014] Specifications and Resources towards a Language for Spatial Computing: Spatial-Ucsb/ConceptsOfSpatialInformation. Haskell, JavaScript, Python, RDF. spatial@ucsb. https://github.com/spatial-ucsb/ConceptsOfSpatialInformation, accessed September 26, 2018.

Vahedi, Behzad, Werner Kuhn, and Andrea Ballatore
2016 Question-Based Spatial Computing—A Case Study. *In* Geospatial Data in a Changing World Pp. 37–50. https://link.springer.com/chapter/10.1007/978-3-319-33783-8_3, accessed February 23, 2017.

Appendix

A. Conventional Analysis

```
"""-----------------------------------------------------------------------------
Name:         Urban Water Analysis
              Conventional analysis
Purpose:      find buildings covered by distance and elevation
Project:      language for spatial computing
Author:       ICRC (2017), adapted by Selina Studer
License:      Apache License 2.0
Created:      26.12.2018
Libraries:    arcpy
-----------------------------------------------------------------------------"""

from arcpy import CheckOutExtension, env, SelectLayerByLocation_management,
        FeatureToPoint_management, CopyFeatures_management, JoinField_management,
        SelectLayerByAttribute_management
from arcpy.sa import ExtractValuesToPoints
from arcpy.da import SearchCursor

env.overwriteOutput = True
CheckOutExtension("Spatial")

# load input data
area = 'C:/area.shp'
dem = 'C:/dem.tif'
waterPoint = 'C:/waterPoints.shp'
building = 'C:/buildings.shp'

# set parameters
distance = 50
elevation = 3

# select water points within area
waterPoint_inArea = SelectLayerByLocation_management(waterPoint, 'INTERSECT', area)

# select buildings within area
building_inArea = SelectLayerByLocation_management(building, 'INTERSECT', area)

# elevation of waterPoints
waterPoint_elev = ExtractValuesToPoints(waterPoint_inArea, dem, 'in_memory/wp_elev')

# calculate elevation of buildings (using centroid)
building_point = FeatureToPoint_management(building_inArea, 'in_memory/building_point', 'CENTROID')
building_pt_elev = ExtractValuesToPoints(building_point, dem, 'in_memory/building_pt_elev')
building_elevation = CopyFeatures_management(building_inArea, 'in_memory/building_elevation')
JoinField_management(building_elevation, 'FID', building_pt_elev, 'FID', 'RASTERVALU')

cursor = SearchCursor(waterPoint_elev, ['OID@', 'SHAPE@', 'RASTERVALU'])
for wp in cursor:
   geom = wp[1]

   # select buildings within distance
   D = str(distance) + ' Meters'
   inDistance = SelectLayerByLocation_management(building_elevation, 'WITHIN_A_DISTANCE', geom, D)

   # select buildings within elevation
   sql = 'RASTERVALU >= ' + str(wp [2] - elevation) + ' AND ' +
   'RASTERVALU <= ' + str(wp [2] + elevation)
   WDWE = SelectLayerByAttribute_management(inDistance, 'SUBSET_SELECTION', sql)
   #  save output
   CopyFeatures_management(WDWE, 'C:/WDWE_' + str(wp [0]) + '.shp')
```

B. Analysis with the Core Concepts

```
"""-----------------------------------------------------------------------------
Name:         Urban Water Analysis
              Analysis with the core concepts
Purpose:      find buildings covered by distance and elevation
Project:      language for spatial computing
Author:       Kuhn et al. 2018, adapted by Selina Studer
License:      Apache License 2.0
Created:      26.12.2018
Libraries:    coreconcepts based on arcpy -----------"""

from coreconcepts.utils import *


# load input data
area = makeObject('C:/area.shp')
dem = makeField('C:/dem.tif').restrictDomain(area, 'inside')
waterPoint = makeObject('C:/waterPoints.shp').restrictDomain(area, 'inside')
building = makeObject('C:/buildings.shp').restrictDomain(area, 'inside')


# set parameters
distance = 50
elevation = 3


# Question 1: What are the elevations of the water points?
waterPoint_elev = waterPoint.addProperty(dem)

# Question 2: What are the elevations of the buildings?
building_elev = building.addProperty(dem)

for wp in waterPoint_elev:

   # Question 3: Which buildings are within the distance of the water point?
   wp_buffer = wp.buffer(distance, 'Meters')
   buildings_in_d = building_elev.restrictDomain(wp_buffer, 'inside')

   # Question 4: Which buildings are within the elevation parameter of the  water point?
   wpElev = wp.get('RASTERVALU')
   sql = 'RASTERVALU >= ' + str(wpElev - elevation) + ' AND ' +
   'RASTERVALU <= ' + str(wpElev + elevation)
   WDWE = buildings_in_d.withProperty(sql)


   # save output
   id = wp.get('FID')
   WDWE.save('C:/out', 'WDWE_' + str(id), '.shp')
```

C. Core Concepts Library

utils.py

```
"""-------------------------------------------------------------------------
Name:         utils.py
Purpose:      coreconcepts library
Project:      language for spatial computing
Author:       Kuhn et al. 2018, adapted by Selina Studer
License:      Apache License 2.0
Created:      26.12.2018
Libraries:    coreconcepts, arcpy -----------"""

# make CcField instance
from coreconcepts.fields import GeoTiffField
from coreconcepts.objects import ArcShpObject
from arcpy import Describe, CopyFeatures_management, SelectLayerByAttribute_management

def makeField(filepath):
    """
    :param filepath: data source file path
    :return: new Ccfield instance
    """
    domain = determine_domain(filepath)

    # determine input file type
    if filepath.endswith(".tif"):
        return GeoTiffField(filepath, id(filepath), domain)
    elif filepath.endswith(".mp3"):
        pass
    assert 0, "Bad shape creation: " + filepath

def makeObject(filepath):
    """
    :param filepath: data source file path
    :return: new Ccobject instance
    """
    domain = determine_domain(filepath)

    # determine input file type
    if filepath.endswith((".shp", "")):             # NOTE:"" for files in_memory or gdb
        return ArcShpObject(filepath, id(filepath), domain)
    elif filepath.endswith(".mp3"):
        pass
    assert 0, "Bad shape creation: " + filepath

def determine_domain(filepath):
    """
    :param filepath: data source filepath
    :return: ArcPy domain extent
    """
    desc = Describe(filepath)
    return desc.extent
```

coreconcepts.py

```
"""-------------------------------------------------------------------------
Name:          coreconcepts.py
Purpose:       coreconcepts library
Project:       language for spatial computing
Author:        Kuhn et al. 2018, adapted by Selina Studer
License:       Apache License 2.0
Created:       26.12.2018
Libraries:     arcpy -----------"""

from arcpy import CheckOutExtension, env

env.overwriteOutput = True

# Check out any necessary licenses
CheckOutExtension("spatial")

class CcField(object):
    """
    Abstract class for core concept 'field'
    """
    def __init__(self, filepath, objIndex, domain):
        """
        :param filepath: data file path
        :param objIndex: unique ID
        :param domain: desc.extent of the geo_object
        """
        self.filepath = filepath
        self.sObj = objIndex
        self.domain = domain

class CcObject(object):
    """
    Abstract class for core concept 'object'
    """
    def __init__(self, filepath, objIndex, domain):
        """
        :param filepath: data file path
        :param objIndex: unique ID
        :param domain: desc.extent of the geo_object
        """
        self.filepath = filepath
        self.sObj = objIndex
        self.domain = domain
```

fields.py

```
"""-------------------------------------------------------------------------
Name:         fields.py
Purpose:      coreconcepts library
Project:      language for spatial computing
Author:       Kuhn et al. 2018, adapted by Selina Studer
License:      Apache License 2.0
Created:      26.12.2018
Libraries:    os, coreconcepts, arcpy -----------"""

import os
from coreconcepts.coreconcepts import CcField
from coreconcepts import utils
from arcpy import Describe, CopyRaster_management
from arcpy.sa import ExtractByMask

class GeoTiffField(CcField):
    """
    Concrete class for core concept 'field'
    For handling .tif files
    """
    def __init__(self, filepath, objIndex, domain):
        super().__init__(filepath, objIndex, domain)
        self.filepath = filepath
        self.sObj = objIndex
        self.domain = domain
        self.filename = os.path.basename(filepath)

    def restrictDomain(self, object, operation):
        """
        Restricts current instance's domain based on object's domain
        @param object: extent to which the field is restricted
        @param operation: valid options: "inside", "outside"
        """

        if operation == 'inside':

            name = "restDom_in_" + str(self.sObj)
            outputLocation = "in_memory\\" + name + ".tif"

            # extract by mask
            outRaster = ExtractByMask(self.filepath, object.filepath)
            CopyRaster_management(outRaster, outputLocation)
            restDom = utils.makeField(outputLocation)

        elif operation == 'outside':
            raise NotImplementedError("restrictDomain 'outside'")

        else:
            raise NotImplementedError(operation)

        # update cc instance's attributes
        desc = Describe(outputLocation)
        restDom.filepath = outputLocation
        restDom.domain = desc.extent
        restDom.filename = os.path.basename(outputLocation)

        return restDom

    def local(self, fields, operation):
        raise NotImplementedError("getValue")

    def coarsen(self, cellW, cellH):
        raise NotImplementedError("getValue")

    def getValue(self, pos):
        raise NotImplementedError("getValue")

    def domain(self):
        return self.domain

    """
    helper methods
    """

    def save (self, Output_Folder, Output_Name, extension):
        outputLocation = Output_Folder + "\\" + Output_Name + extension
        print("saved to", outputLocation)
        CopyRaster_management(self.filepath, outputLocation)
```

objects.py

```
"""_____________________________________________________________________________
Name:         objects.py
Purpose:      coreconcepts library
Project:      language for spatial computing
Author:       Kuhn et al. 2018, adapted by Selina Studer
License:      Apache License 2.0
Created:      26.12.2018
Libraries:    os, coreconcepts, arcpy -----------"""

import os
from coreconcepts.coreconcepts import CcObject
from coreconcepts import utils
from arcpy import Buffer_analysis, Describe, Delete_management, CopyFeatures_management, ListFields,
FeatureToPoint_management, JoinField_management, SelectLayerByLocation_management, SelectLayerByAt-
tribute_management, MakeFeatureLayer_management
from arcpy.sa import ExtractValuesToPoints
from arcpy.da import SearchCursor

class ArcShpObject(CcObject):
    """
    Concrete class for core concept 'object'
    For handling .shp files and feature classes of a geodatabase
    """

    def __init__(self, filepath, objIndex, domain):
        super().__init__(filepath, objIndex, domain)
        self.filepath = filepath
        self.sObj = objIndex
        self.domain = domain
        self.filename = os.path.basename(filepath)
        self.OIDs = SearchCursor(self.filepath, "OID@")

    def __iter__(self):
        return self

    def __next__(self):
        try:
            next_FID = next(self.OIDs)[0]
        except StopIteration:
            self.OIDs = SearchCursor(self.filepath, "OID@")
            raise StopIteration
        next_filepath = f"{self.filepath}_FID={next_FID}"
        MakeFeatureLayer_management(self.filepath, next_filepath)
        SelectLayerByAttribute_management(next_filepath, "NEW_SELECTION", f"FID={next_FID}")
        return ArcShpObject(next_filepath, id(next_filepath), self.domain)

    def buffer (self, distance, unitType ):
        """
        Buffer input object
        @param distance: buffer distance
        @param unitType: unit type
        """

        # determine temporary unique file
        distName = str(distance)
        distName2 = distName.replace(".", "_")
        print("distName2", distName2)
        name = "buf_" + str(self.sObj) + distName2
        outputLocation = "in_memory\\" + name

        # calculate buffer
        concatDistance = str(distance) + " " + unitType
        Buffer_analysis(self.filepath, outputLocation, concatDistance)
        bufObj = utils.makeObject(outputLocation)

        # update cc instance's attributes
        desc = Describe(outputLocation)
        bufObj.domain = desc.extent
        bufObj.filepath = outputLocation
        bufObj.filename = os.path.basename(outputLocation)

        return bufObj

    def restrictDomain(self, object, operation):
        """
        Restricts current instance's domain based on object's domain
        @param object: extent to which the object is restricted
        @param operation: valid options: "inside", "outside"
```

```
        """

        name = "restDom_" + str(self.sObj)
        outputLocation = "in_memory\\" + name

        if operation == 'inside':
            # select by location
            select = SelectLayerByLocation_management(self.filepath, "INTERSECT", object.filepath)
            CopyFeatures_management(select, outputLocation)
            restDom = utils.makeObject(outputLocation)

        elif operation == 'outside':
            # select by location
            sel = SelectLayerByLocation_management(self.filepath, "INTERSECT", object.filepath)
            select = SelectLayerByLocation_management(sel, "INTERSECT", object.filepath, "",
            "SWITCH_SELECTION")
            CopyFeatures_management(select, outputLocation)
            restDom = utils.makeObject(outputLocation)

        else:
            raise NotImplementedError(operation)

        # update cc instance's attributes
        desc = Describe(outputLocation)
        restDom.domain = desc.extent
        restDom.filepath = outputLocation
        restDom.filename = os.path.basename(outputLocation)

        return restDom

    def get(self, prop):
        """
        :param: name of the property
        :returns: value of property in the object
        """

        with SearchCursor(self.filepath, prop) as cursor:
            for row in cursor:
                return row[0]

    def addProperty(self, in_raster):
        """
        get value of a field and write it to a column named RASTERVALU in the object
        @param in_raster: raster where the value is taken from
        """

        desc = Describe(self.filepath)
        name = "addProperty" + str(self.sObj)
        outputLocation = "in_memory\\" + name

        if desc.shapeType == "Point":
            ExtractValuesToPoints(self.filepath, in_raster.filepath, outputLocation)
            addProperty = utils.makeObject(outputLocation)

        elif desc.shapeType == "Line":
            raise NotImplementedError(desc.shapeType)

        elif desc.shapeType == "Polygon":
            polyToPoint = "in_memory\\polyToPoint_" + str(self.sObj)
            FeatureToPoint_management(self.filepath, polyToPoint, "CENTROID")
            valueToPoint = "in_memory\\valueToPoint_" + str(self.sObj)
            ExtractValuesToPoints(polyToPoint, in_raster.filepath, valueToPoint)
            CopyFeatures_management(self.filepath, outputLocation)
            JoinField_management(outputLocation, "FID", valueToPoint, "FID", "RASTERVALU")
            addProperty = utils.makeObject(outputLocation)
            Delete_management(polyToPoint)
            Delete_management(valueToPoint)
            # TODO: implement method that the parameters "CENTROID" or "INSIDE" for
            FeatureToPoint_management() can be selected

        else:
            raise NotImplementedError("unknown shapeType:", desc.shapeType)

        # update cc instance's attributes
        desc = Describe(outputLocation)
        addProperty.domain = desc.extent
        addProperty.filepath = outputLocation
        addProperty.filename = os.path.basename(outputLocation)

        return addProperty

    def withProperty(self, sql):
```

```
        """
        :param sql: sql expression
        :returns: feature that meets the properties of the sql expression
        """

        name = "wProp_" + str(self.sObj)
        outputLocation = "in_memory\\" + name

        selByAtt = SelectLayerByAttribute_management(self.filepath, "NEW_SELECTION", sql)
        CopyFeatures_management(selByAtt, outputLocation)
        wProp = utils.makeObject(outputLocation)

        # update cc instance's attributes
        desc = Describe(outputLocation)
        wProp.domain = desc.extent
        wProp.filepath = outputLocation
        wProp.filename = os.path.basename(outputLocation)

        return wProp

    """
        helper methods
    """

    def save(self, Output_Folder, Output_Name, extension):
        outputLocation = Output_Folder + "\\" + Output_Name + extension
        CopyFeatures_management(self.filepath, outputLocation)

    def show(self):
        print("\n")
        print("show 5 first table rows for file:", '\x1b[1;36m' + self.filepath + '\x1b[0m')
        list = []
        fields = ListFields(self.filepath)
        for field in fields:
            list.append(field.name)

        list.remove("Shape")
        header = []
        for field in list:
            header.append(str('{:_^20}'.format(field)))
        print(header)

        count = 1
        with SearchCursor(self.filepath, list) as cursor:

            line = []
            for row in cursor:
                for col in row:
                    line.append(str('{:^20}'.format(col)))
                print(line)
                line = []
                if count >= 5:
                    break
                count += 1

        del cursor
```