A lightweight web client to discover, explore and test software components that process geographical data

Bénédicte Bucher

A lightweight web client to discover, explore and test software components that process geographical data

Bénédicte Bucher

COGIT Laboratory, Institut Géographique National,
2 av. Pasteur, 94160 Saint Mandé, France
benedicte.bucher@ign.fr

1. Introduction

Processing geographical data based on modular software components developed by different authors is getting more and more generalised with web services and interoperable java libraries. For instance, MapShaper and WebGen are generalisation Web services meant to be invocated by human users on their data thanks to clients hiding the complexity of network communication [1] [2]. Geotools and Geoxygene are java libraries, that can be downloaded on the Web, and which propose a wide range of functionality related to geographical data processing. These are interoperable because they implement ISO/OGC concepts like FT_Feature or GM_Point, and, in the future, because they will implement the GeoAPI (a set of standard java interface definitions corresponding to these conceptual models).

This paper presents a work in progress to provide a lightweight web interface for a user to locate software components that process geographical data, to explore them and eventually test them on his data. The immediate purpose of this application is to facilitate the design of user applications based on such resources. A user may want to select a best implementation of an abstract service before deciding to use a web service or to use the programmatic interface of a java interoperable library that has not yet been deployed as a web service. The expression 'abstract service' is used here to refer to 'an abstract resource that represents a capability of performing tasks that represents a coherent functionality' [3].

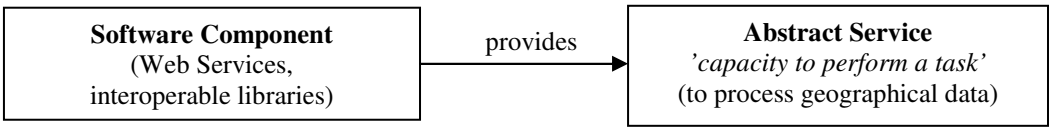

Figure 1. The aim of this work is to support the discovery, exploration and exploitation of software components that provide abstract services. We focus on services of the kind 'to process geographical data'.

Another purpose of this work is to stimulate developers of java libraries to publish methods that are relevant outside the library. There exist environments to deploy java methods as web services like the axis application. They cop with network communication –incl. XML serialisation of main datatypes- and with the writing of standard metadata. They do not assist the developer in identifying relevant methods, in choosing meaningful names for the service and for the parameters, in defining the best granularity for his service operations. We think that our application will incite them to describe abstract services provided by their libraries, which is a first step towards deploying them as web services if necessary. For this purpose, an interface to author metadata about services provided by libraries is being developed: the model for these metadata is an ad hoc model that extrapolates standard models for Web Services [4]. This interface is not presented in this paper.

In the following of this paper, we adapt principles stemming from spatial data infrastructure literature like [5] and from search engines to our context: user access to software components that process geographical data. We define several access phases as schematised on figure 2: discovery, exploration, and testing. For each phase, a short analysis of its functional requirements is followed by the description of our proposal to support them.

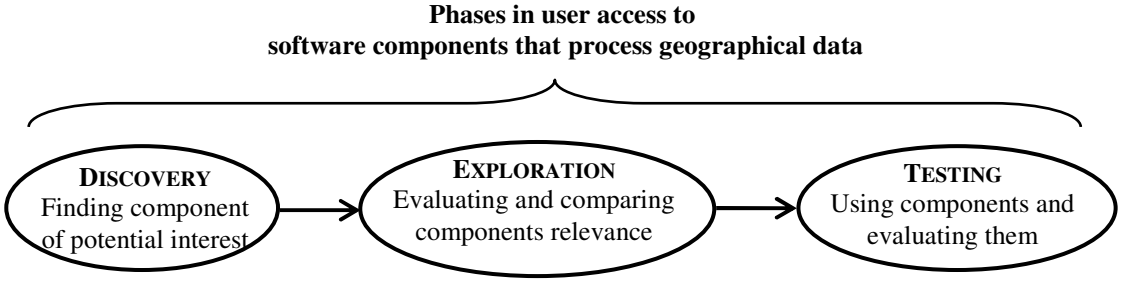

Figure 2. Main phases in user access to software components that process geographical data.

The prototype implementation is a signed java applet that communicates with servlets.

2. Discovering relevant software components that process geographical data

The discovery phase aims at defining, among a set of resources (like all items of a catalogue or the whole Web), a first subset of resources that are potentially relevant with respect to the user query. Two important aspects in discovery are supporting the expression of the user need and selecting relevant resources among a possibly vast set. These reasons

induce several requirements on the layer of data used in discovery. Discovery data should contain meaningful criteria of relevance about the resources (like the spatial extent of a geographical data set or the words contained in a Web document). These criteria should make sense as homogeneously as possible among the resources (for instance, spatial extent is not a homogeneous criteria for Web documents). These data should be organised into a queriable structure (like a SQL database or inverted files). Last, these data should reference the resources.

The main standard metadata model for cataloguing web services is UDDI [6]. It describes a service through four main metadata entities: the businessEntity describes the provider, the businessService describes the type of service, the bindingTemplate and tModel contain information about how to bind to the service and about the service formats. Discovery UDDI criteria are the 'BusinessEntity' and the 'BusinessService'. Each of them is specified through a bag of categories in taxonomies. Publication of taxonomies is supported by tModels like the Universal Standard Products and Services Classification (UNSPSC) UDDI tModel. In the following we concentrate on the yellow pages criterion, namely the service category.

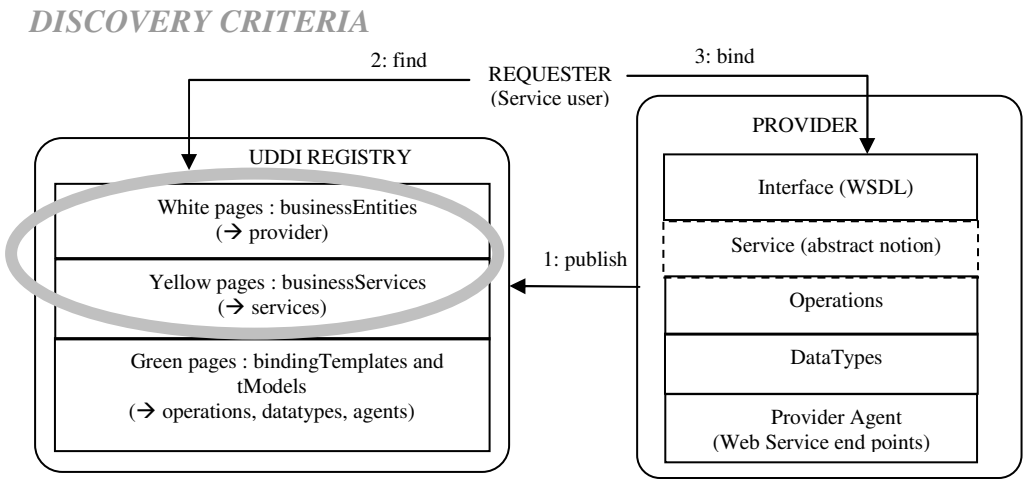

Figure 3. Registry big picture: providers publish their services by documenting specific metadata in a registry. A requester finds a service thanks to the registry metadata. A requester binds to the provider to use a service he has found.

There exist taxonomies for the resources we are studying. The ISO 19119 service taxonomy identifies very high level categories of GI services among which 'geographic processing services'. This category is divided into four subcategories after the nature of the processed information ('spatial', 'thematic', 'temporal', 'metadata'). Each ISO19119

subcategory is specialised with a list of services, like ‘generalisation services’. This taxonomy is non exhaustive and still too generic. For instance, [2] propose more specific classifications of ‘generalisation services’. One is obtained based on the service data types. Services which data types may be expressed with GML core schema elements are classified into ‘default generalisation services’ and service which data types need more GML application schema elements are classified into ‘advanced generalisation services’. These illustrate the importance of signatures in services classification. The word signature refers here to the data types of input and output parameters of the service. Another classification of generalization services proposed by [2] is based on generalisation activities: generalization support services which prepare the data, generalization operator services which transform the data, generalization process services which control the whole generalization task. These illustrate the importance of other relationships between services than subsumption: the sequencing relationship (this service is to be used before that other one), the composition relationship (this service uses that other one). When modelling services categories, other challenges are to describe services unambiguously and also to explicit relationships between services in a tractable way. [7] addresses this by proposing an operations ontology OPERA expressed in the formal language OWL. Earlier work to classify operations took place in the context of formalising the use of GIS to perform spatial analysis. [8] and [9] have formalised abstract spatial analysis operations and combination of operations in their map algebra. Later, [10] has proposed a list of universal abstract operations that refer to GIS operations. His VGIS (Virtual GIS) prototype allows the user to build sequences of abstract operations in the form of flow charts. The system interprets user flow charts into executable processes. [11] have added to VGIS hybrid operations on vector and field data. Other authors concentrated on higher level functionality. For instance, [12] focuses on the functionality ‘portraying thematic data’ and proposes an explicit strategy to perform this functionality with a specific GIS. Such expert knowledge about processing geographical data could be integrated into a model of abstract services to facilitate the discovery of software components based on user goals [13].

To summarize, there exist several classifications or more complex models for abstract services ranking from elementary operations to complex processes. Supporting user discovery of services of interest requires integrating categories identified in these models into a unified model. Whereas many models already address the discovery of new services based on the sequencing of existing services –services choregraphy models-, a first result

not yet achieved is a model that addresses user discovery of existing services, possibly predefined sequences of services –services orchestration models-. Important criteria for user discovery are the service name, the service signature as well as subsumption and composition relationships between services.

We propose an ad hoc model summarised on figure 3. In this model, the word 'function' is used instead of 'service'. A function is 'what the component does'. It has several properties among which a generic name (which should refer to an existing classification) and a unique identifier (the name is not an identifier). As in standard models for web services, function signature is represented through parameters (variables which include geographical data input and output), preconditions and postconditions. The formalisation of composition relationships, i.e. of methods and strategies, is classically done either with state diagrams or with activity diagrams. In our model, we chose the latter [14]. Activities associated to a function are descriptions of how to carry out this function. It can be a sequence of message to be exchanged with a web service (this sequence is described in standard metadata for the web service (WSDL, OWL-S)) or it can be a sequence of method calls from a library.

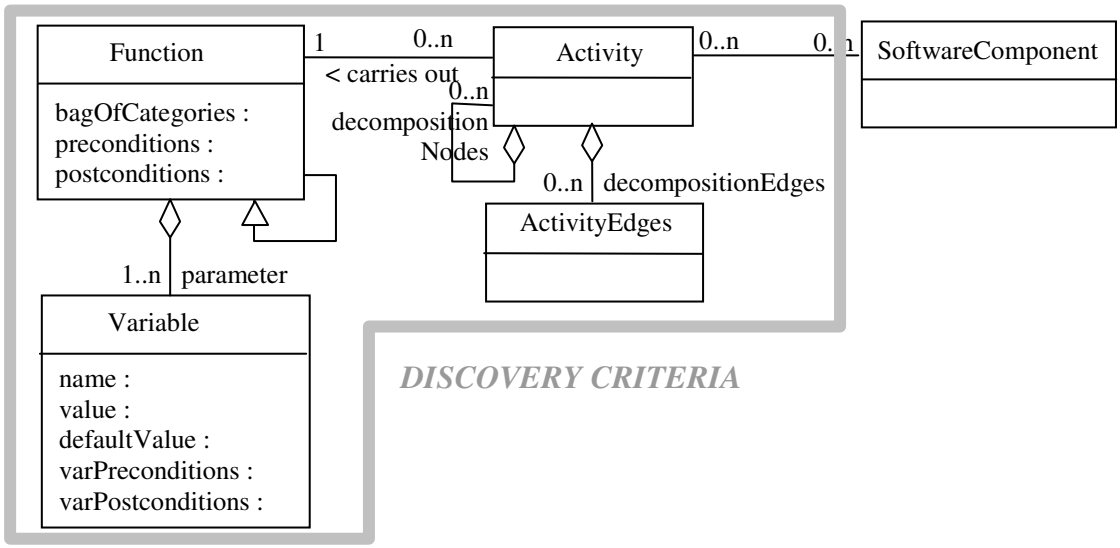

Figure 4. Main lines of the model to index software components by the function they provide and to support function discovery based on signatures, subsumption and composition relationships.

In the prototype, users can browse this model based on a tree structure that is progressively sent to the client. In this tree structure, a function node (a service) can have the following sons: parameters, more specific functions, 'may use' functions, software components and activities. An activity node can have the following sons: parameters, sub-

activities, software components. The strategy to send the structure progressively is so far very generic. When the user selects a node, the client sends a request to get the n+3 nodes.

3. Exploring selected resources

Exploration aims at assessing more precisely the relevance of resources selected during discovery and at comparing the relevance of several resources, based on metadata only. Classically, during exploration, a user firstly identifies selected resources thanks to identification metadata (like URLs titles). Relevance assessment is then supported by the display of a relevance criterion for each resource (like fragments of a document containing the words used in the query). Last, the user also needs to compare relevance and determine the most relevant resources. He may need help to do this, for instance when there are too numerous selected resources for him to browse all relevance criteria. Automatic comparison consists in clustering (grouping resources that are relevant the same way) and ranking (ordering resources after their level of relevance). It needs tractable comparison criteria (like the frequency of the user keyword in the document or a URL's popularity). In the following of this section, we try to support the identification, relevance assessment and comparison of specific resources: software components that process geographical data.

Identification based on a cartographical illustration of provided functions

In our context, the user needs to identify two items: the function and the component. Component identification can first rely on web services URLs and java packages names. Function identification is a complex issue as long as there does not exist a detailed universal ontology of functions involved in the processing of geographical data. When a function is cartographic or modifies the geometry, a graphical overview will enhance its identification by the user. In [4], the developer of a processing tool can add a graphical illustration to the tool description. He may generate this illustration by processing sample data stored on the server and by portraying the processed data with styles proposed by the server.

A finer illustration of a function may be a mosaic of results corresponding to different parameter values and to different input data. Indeed, for some processes, differences in parameter values or difference in properties of input data yield very different results. This issue has been explored in [15]. The authors have built a database of cartographic samples obtained by applying the same generalisation algorithm with different parameter values and to different geographical data. This allows illustrating the various behaviours of this

algorithm depending on parameter values and on properties relative to the information represented in input data (object size and granularity, spatial structure). Other properties of the input data may affect the behaviour of a software: properties relative to the global data set (size of the data set) or relative to the data logical structure (attribute values).

Relevance assessment based on activities

The relevance of a component to provide a service should include effectiveness and efficiency criteria. We focus on efficiency criteria. In our model (illustrated on figure 4), activities describe sequences of actions that should be undertaken to carry out functions thanks to components. Assessing the relevance of a component with respect to a selected function relies on properties of the activity 'to carry out the selected function thanks to the component'. These properties are: the number of actions, the number of extra software components, the existence of default value for parameters, the complexity of types involved in the signature. We do not calculate tractable relevance criteria but display these properties for the user to browse them.

4. Testing selected resources

Last, the user needs to test the resources on his data. More precisely, the user needs to upload his data on the server to have them processed. He needs to test a selected function to his data. To do so, he must value the parameters of an activity that carries out this function. When the required parameters are valued, the activity state becomes 'executable' and the user can ask the server to execute it. He needs to see the result, even if his processed data remain on the server. The user also needs to step back to a former state of his data, try another parameter value or another activity (another sequence of methods calls or web services calls).

Specifying activities

User specification of activities is based on the 'obrowser' java component developed at the laboratory and embedded in our applet. This component builds a graphical view of any java object (based on his class definition) for the user to interactively specify it. 'Obrowser' can either provide a generic view –for any class definition on the classpath- or a view dedicated to the current class –thanks to an extension mechanism-. This approach is based on the assumption that function parameters can be expressed as instances of java

classes. When the function is based on web services, the parameter are ultimately XML fragments but they can be mapped to java objects. The corresponding java class are obtained by applying a stub generator tool to the service WSDL metadata. In our context, genericity is important because function parameters can be of any type.

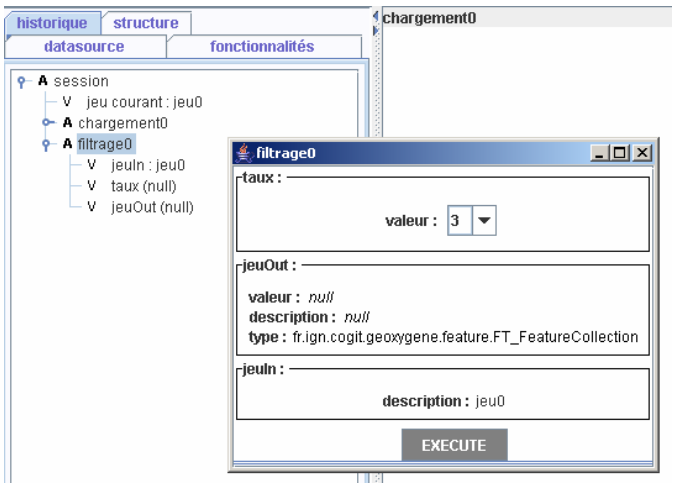

Figure 5. Documenting the value of an elementary activity: to carry out a filtering.

Distributing session actions and session data between the client and the server

Stepping back and trying another activity is supported by tracing all user and server actions on geographical data and keeping a copy of data states. This tracing is stored in the 'Session History' which is formalised as an Activity under construction. Each time the user wants to carry out a new function, a new sub-activity edge is added to the History as well as an edge. This subactivity is created as the carrying out of this function. The subactivities may have several states: specifiable, executable, executed. A crucial aspect of a History is that some parameter values are documented through identifiers instead of the very values. These identifiers are keys belonging to the ServerTable and the ClientTable. The ServerTable contains FeatureCollection and other complex objects related to the Geoxygene plateform. It is not shared with the client. The ClientTable contains SVG documents. When the user uploads his data to the server, the server generates a SVG document and sends it to the client along with an identifier for that document. It also indexes the FeatureCollection with the same identifier in the ServerTable attached to the user session. The client receives the SVG document and indexes it in his ClientTable, with the identifier. The History is modified on the client each time the user selects a new function, or specifies a sub-activity. It is modified on the server each time the server runs

an executable sub-activity. In all cases, the whole history is sent to the client or server to replace the older one. When user data are a large data set, the server does not work on the whole data set but on an extract of them. This extract is made automatically at loading time.

SVG activity maps: a library of styles dedicated to render process effects

Visualisation of a process result is based on two elements. First element is a library of styles available on SVG maps dedicated to portray executed activities. These SVG maps are called 'SVG activity maps'. Currently studied styles are limited to rendering 'feature modifications'. The user specifies the style symbol and the server classifies features into graphical classes (classes of features to be drawn with a given style). This server process is based on identifiers for the features that are added to the feature collection before executing an activity. So far, only three generic styles are provided: 'deleted feature', 'feature with modified literal attributes', 'created feature'. The 'created feature' style is specified into 'created geometry' and 'created class'.

'Created geometry' is a style adapted to render buffer. It is calculated based on default styles adapted to any type of geometry, and on the color of input data. It is a dynamic style. The user can modify the opacity balance rate between the created feature and the other cartographic layers: when the created geometry layer is opaque the other layers are transparent, and vice versa.

'Created class' is a style adapted to render the association of features, typically through the creation of a classification attribute or the creation of relationships between features.

Complex cartographic views based on SVG activity maps

Next element is a cartographic view attached to the following history nodes contents: variables, executed activities and 'decomposition'. The view attached to an executed activity is a SVG activity map with the style presented in the preceding section. The view attached to a 'decomposition' property or to the root is a set of SVG activity maps corresponding to all subactivities. Activity maps can be displayed in a chronological layout or in a workflow.

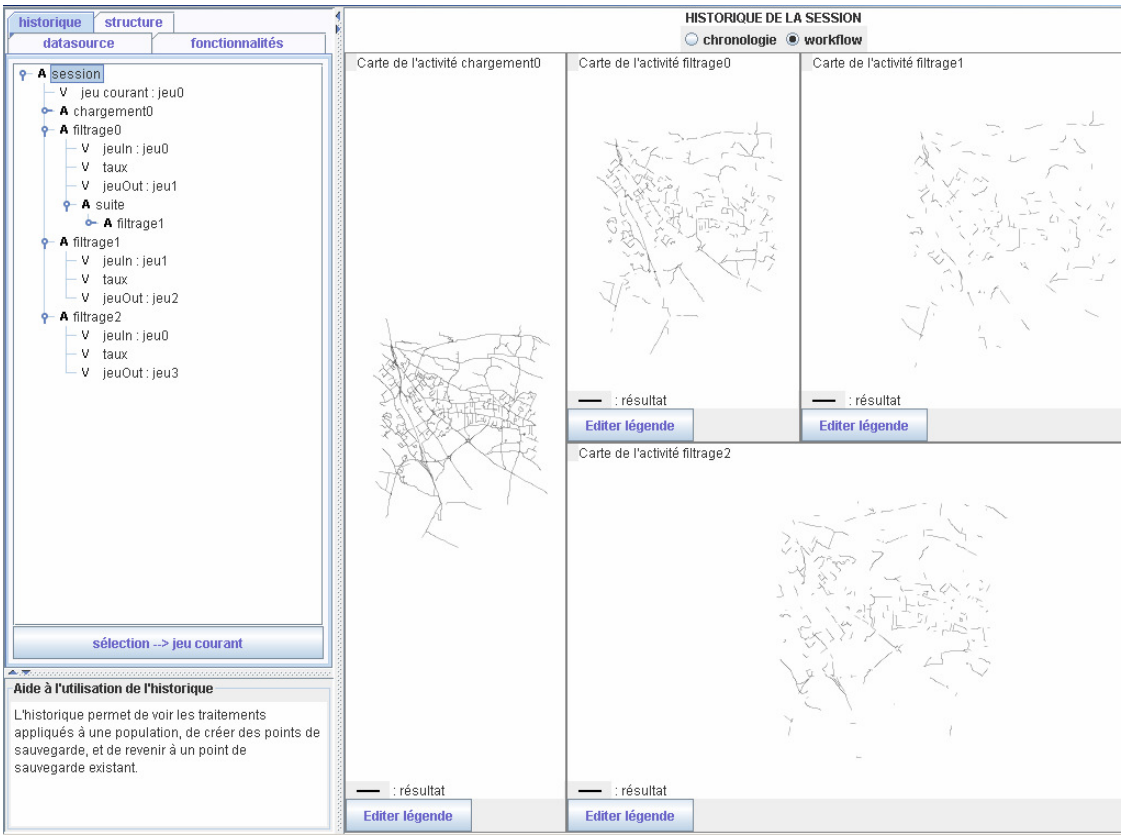

Figure 6. A workflow display of a session : the user firstly loaded his data on the server (map on the left), then applied successively two filtering to the data set (two maps above), stepped back to the initial loaded data and applied another filtering with a different parameter value (map below).

The view attached to a variable is the set of SVG maps corresponding to activities that count the value of this variable among their own variables. For instance, when the user clicks on a variable which value is a data set, the client identifies the activity that yields this data set and the activities that transform it. The user may specify a legend for the data set in one of the SVG canvas and ask the server to generate SVG documents with the same style for the datasets displayed in the other SVG canvas of the current view. So far, he can specify some basic SVG attributes that are mapped to styling rules.

5. Conclusion

The work presented in this paper addresses the issues of discovering software components that process geographical data, assessing their relevance and testing them from a lightweight web client embedded in a classical browser. It is grounded firstly on a model that relates functions to components thanks to activities. Activities describe how to carry out functions with components. It is also grounded on styling tools to render the result of

one activity on a SVG activity map, and to render a sequence of tests on complex views. Our prototype supports the specification and launching of a complex process to be executed on a remote server. The user can get SVG feed backs of process results.

This prototype aims at refining and testing our approach. Yet, to propose an application that meets real needs we must enrich it with more components and more activities.

References

1- Harrower, M., Bloch, M., MapShaper.org: A Map Generalization Web Service, in IEEE Computer Graphics and Applications, 2006, vol 26(4), pp.22-27

2 - Burghardt, D., Neun, M. and Weibel, R., Generalization Services on the Web - A Classification and an Initial Prototype Implementation, in CaGIS, 2005, Vol. 32, No. 4

3 - W3C working group, Web Services Architecture, W3C Note, 2004

4 - Abd El Kader Y., Bucher B., 2006, Cataloguing GI Functions provided by Non Web Services Software resources Within IGN, in proceedings of the AGILE conference, Visegrad, 2006 (apendum)

5 - GSDI, Developing Spatial Data Infrastructures: the SDI Cookbook, Douglas D. Nerbert (Ed), v2.0, 2004

6 - UDDI Spec Tec, UDDI Version 3.0.2, 2004

7 - Lemmens, R., Semantic interoperability in distributed geo-service, PhD thesis, ITC, Enschede, 2006

8 - Joseph Berry, Fundamental Operations in Computer-Assisted Map Analysis, IJGIS, Vol 1, No2, 1987, pp119-136

9 - Tomlin C. D., Cartographic Modelling, in *Geographical Information Systems : Principles and Application*, edited by D. J. Maguire, M. F. Goodchild and D. Rhind (Harlows : Longmans), 1991, vol1, pp.361-374

10 - Albrecht J., Universal GIS operations for environmental modeling, in Proceedings of the 3rd International Conference on Integrating GIS and Environmental Modeling, Santa Barbara, 1996

11 - Jung S., Voser S. A., Ehlers M., Hybrid Spatial Analysis Operations as a Foundation for Integrated GIS, in proceedings of the ISPRS Commission IV Symposium, Stuttgart, Germany, 1998

12 - Andy Mitchell, *The ESRI Guide to GIS Analysis,* volume 1 : Geographic patterns & relationships, USA, 1999

13 - Bucher B., Translating user needs for geographic information into metadata queries, in proceedings of the 6th AGILE conference, Lyon, 2003, pp. 567-576

14 - Bucher B., Balley S., Richard D., Cébelieu G., Hangouët J-F. Shareable descriptions of data production processes, in proceedings of the 8th AGILE conference, Estoril, Portugal, 2005

15 - Hubert F., Ruas A., A method based on samples to capture user needs for generalisation, 5th Workshop on Progress in Automated Map Generalization, Paris, France, 2003