Cataloguing GI Functions provided by Non Web Services Software Resources Within IGN

Yann Abd-el-Kader, Bénédicte Bucher
Laboratoire COGIT – Institut Géographique National –
2 av Pasteur – 94 165 Saint Mandé Cedex – France
{yann.abd-el-kader,benedicte.bucher}@ign.fr

SUMMARY
Numerous software resources are available within IGN, the French National Mapping Agency, to manipulate geographical data. These are commercial off-the-shelves software or ad hoc tools developed within IGN. Sharing these resources and information about them is necessary but is still too difficult. This paper presents our approach to catalogue resources available within IGN to manipulate geodata. We extend upon OWL-S metadata model for Web Services to propose a metadata model that describes services provided by any software resource that is not a Web Service. A crucial aspect of this extension is the modification of the metadata entity called 'grounding'. The grounding of a service in OWL-S is mostly the description of the sequence of XML messages that must be send to or received from the Web Service URL to achieve this service. In our context, a service 'grounding' is described as a sequence of more generic steps. Besides, it is necessary to adapt a grounding description to the user context.

KEYWORDS : ***Service, Usability, Grounding, Semantic Web***

INTRODUCTION

Discussion lists within IGN account for important needs related to manipulating geodata :

- getting to know if a software resource exists that performs a specific function,
- getting to know if an algorithm exists that performs a specific function,
- getting information about a specific function or a specific software like how to use them or how to interpret the result.

Eventually, people that need to manipulate geographical data sometimes neither correctly use, nor use at all existing software resources that are best fitted to their needs. Many factors contribute to this situation in the field of geographic information. Firstly, available software resources to manipulate geographical data are a fast evolving set. There are many reasons to that. Software in this area are very complex and can always be improved so that there is an everlasting release of new versions of commercial software. The dynamism of free software communities and the revolution brought to geographical data manipulation by interoperability has led to a growing number of free programs to manipulate geographical data. Second, many of these software resources require a long learning period to get to know their potential. Many also require a specific expertness to use them.

Important clues to solve this issue stem from the MDA implementation paradigm. This paradigm leads to the implementation of software resources that are more and more shareable and extendable, through components providing core functionalities and Web Services providing an interface to access these functionalities. The objective of the research presented in this paper is to support the 'a posteriori' cataloguing and sharing of available software resources to manipulate geodata. In other words, we do not develop new software components but we intend to describe existing software that has not yet been made accessible via Web Services.

OUR APPROACH

To describe ‘data manipulation resources’ as if they were Web Services

The development of Web Services to diffuse existing business components follows three main steps :

- Identifying abstract services that are supported by the business components.
- Developing a Web application, called a Web Service, that duplicate or invocate these business components methods.
- Describing how the abstract services can be realized through sequences of XML message exchange between an end user and the Web Service application.

Our approach is to imitate the first and last steps. Thus we distinguish two types of “data manipulation resources”:

- A function is, in this paper, an available process that takes geographical digital data as an input and produces an output, like a buffer or an import. It corresponds to ‘abstract services’ of Web Services. We use the term function instead of process or task or activity to denote a process where the user has only simple actions to control the process and no complex thinking.
- A software resource is everything that is organized into digital files and registries and that can be actually transferred from one computer to another. It may be for instance a java class or a library. Software resources encapsulate functions, like a java class encapsulates methods.

We list hereafter some crucial problems encountered in developing a catalogue of software resources and of functions provided by them.

- Most functions encapsulated in software resources are not identified as resources as such. There does not exist an index for these functions, a place where they are all listed.
- It is difficult to describe a function, to explain what it actually does. Applying a function to a geographical data set often has effects and side effects difficult to describe and to predict.
- Access to functions encapsulated in software that is not a Web Service can not be described as simply as access to a service provided by a Web Service.
- Using a geographic function will often require adapting the user data to the function input structure. It will also require parameterization, which might be difficult when the implemented function relies on a complex algorithm.
- Last, using a geographic function can include modifying the resource itself to adapt it to the user context. And modifying a function is difficult if the developer did not foresee that his code would be reused and modified

Data manipulation resources to be catalogued

A preliminary step in our approach is the definition of a resource model, i.e. to define the types of elements that need sharing or describing. There are three types of resources : algorithms, functions and software resources.

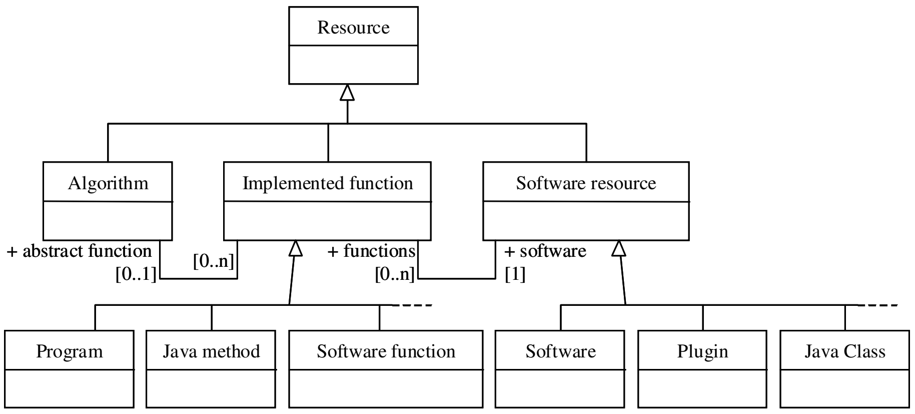

Figure 1: Types of resources that are indexed in the catalogue.

Sometimes the function is itself a software resource, for instance a java class with a main method that processes geographical data. Sometimes it is not, for instance the buffer function in most GIS software. This model is schematized on Figure 1.

Resources must come with a system of unique identifiers. In java, there is already a system of identifiers for implemented functions and software resources. For instance, let us consider the description of a specific DTM warping function that is implemented in a java method. It is identified by its name “public void mou.ChampMNT.ChampMNT.active()”. The software resource that encapsulates this function is the java class identified by its name “mou.ChampMNT”. For resources that are neither java classes nor java methods, domain namespaces have to be established to build such identifiers. We did not impose a structure for these namespace and leave it up to authors to define these namespaces.

There are relationships between resources that are not represented on **Hiba! A hivatkozási forrás nem található.**. A resource may be composed of other resources, like a library is composed of class, a software configuration is composed of a core and plugins. A resource may refer to another resource, like an algorithm that is a variation of another algorithm.

Metadata model

The next step is the definition of a metadata model to describe these resources. There does not exist a metadata model to describe any software and implemented functions. Models dedicated to software design describe software resources and functions. However the description exists prior to the implementation of the function, sometimes prior to the implementation of the software itself, and rules this implementation. Metadata models dedicated to Web services like ISO19119 and WSDL (ISO, 2001)(W3C 2001) do describe already implemented functions. But the resources we describe are not Web Services and many of them don’t have a well defined interface to interact with them, i.e. to specify input values and to get output values. The OWL-S model is yet interesting because it adds other fields to the WSDL description (OWL, 2004). It is organised into three facets : the service *profile* or “what the service does”, the service *model* or “how it does it”, the service *grounding* or “how to use it” which is the WSDL description.

We propose a metadata model dedicated to our needs, i.e. describing functions that are encapsulated into various software resources and that take geographical data as an input. This model is organised into entities schematised on Figure 2: Some entities are directly inspired from the OWL-S model: profile, model, grounding. The evaluation facet records user feed back about their usage of the function. Besides, the software resource itself is described by these four facets.

The detailed structure of these entities varies depending on the type of the resource –algorithm, function or software resource- and on other properties of the resource. Families may be defined to group resources that share the same detailed structure of description, like the families "generalisation measures", "ESRI software" or "Eclipse plugin".

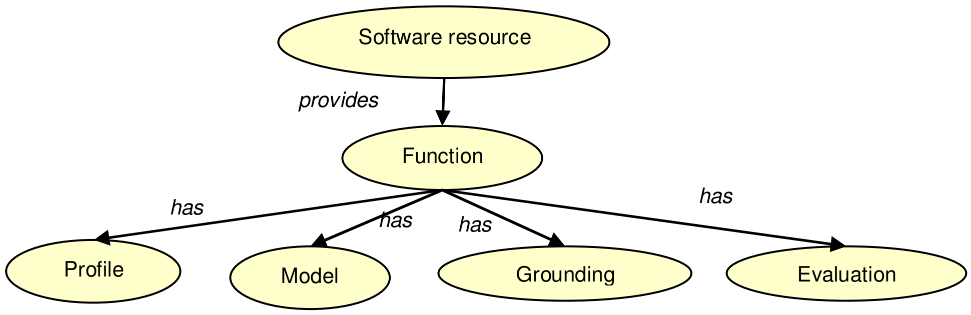

Figure 2: Main classes of our metadata model, directly inspired from OWL-S model. The software resource itself is also described by these four facets.

We describe below some fields that are especially important in this model.

The 'profile' metadata entity

In the entity profile, special attention has been paid to describing inputs and outputs to adapt to the domain of geographical information. Inputs, parameters and outputs are described as variables. A variable has a name that is used to refer to it in the resource description. The value of a variable is described at several interpretation levels :

- the format, e.g. gml or a java object
- the logical type, e.g. a specific data structure that may be written in a gml schema or in a java class,
- the conceptual type,
- the information content.

When the value of the variable corresponds to a type of objects more specific than existing types, for instance "a network that has no dead ends longer than 500m", these properties are described in the elements "variable.precondition" in a way similar to (Lutz, 2005).

A variable may have a structure of its own. For instance, in the example above, the distorsion sources of the DTM are points (point of the DTM) associated to vectors (move of the point). The author of the description may have defined a specific logical type to structure this input (or output). If he has not, he may use the element "variable.precondition" to describe the structure.

Last, there may exist dependency relationships between several variables in a description. For instance let us consider the DTM warping function introduced before which inputs are : the DTM itself, the elasticity coefficients of the DTM and the distorsion sources. The distorsion sources are points that are on the DTM. Such dependencies are described in the element "profile.precondition".

In the entity profile, a field "illustration" has been added. It contains cartographical samples that illustrate the effects of the function. Cartographical samples are built based on a set of data samples.

The 'grounding' metadata entity

If the resource is a software, the grounding consists in reference to basic functions within this software like launching it, loading data and so on.

If the resource is a function, the grounding will describe either how to use it (specify it, run it, interact with it and read its result) or how to reuse it (modify the source code or use an extension point). Steps are actually of the same nature as 'activities'. As such, a step may be composed of substeps. For instance "to display the color chooser" is composed of more elementary GUI steps describing which menu item to use. A step may be described in several ways :

- a programmatic action: *Log on this, Write this,*
- an action on a graphical user interface: *click on this*
- a reference to a function : *Use function F*
- a reference to a goal : Your *data on h:/*

Two important aspects of a function grounding are firstly that a same resource may have several groundings, typically a user interface grounding and a programmatic grounding, and second that there are families of grounding. For instance, functions provided by a non static java method have a programmatic grounding of the following form: import package, create invoker, document relevant variables, invocate method, read result.

In our example, the function "DTM warping" has two groundings. The first one relies on a specific java class that the developer wrote to test the function. It is described as follows :

- To write down the DTM values (input1) in the java method : "public FrameMou.FrameMou".
- To write down the elasticity coefficients (input3) in the java method : "public FrameMou.FrameMou".
- To launch FrameMou main method.
- To mouse-click on points of the displayed DTM (input2.points).
- To mouse-drag (input2.vector).

The second grounding is entirely programmatic.

- To create manually an instance of "ChampMNT", as for instance it is done in the java code : "public FrameMou.FrameMou" (input1 and 3).
- To modify manually the coordinates of one or more nodes of this ChampMNT object (the DTM gets unstable).
- To call the method "public void ChampMNT.active()" on this object.

This example shows that the signature of an implemented function, here "public void mou.ChampMNT.activeChamp()" does not always correspond to its effective inputs and outputs, here the object ChampMNT, its elasticity coefficients and the distorsion sources. These inputs and outputs are defined as class fields in the java class definition mou.ChampMNT so that they don't appear in the method signature.

PROTOTYPE

A prototype has been developed to edit and browse descriptions structured after this model.

Concerning the edition of descriptions, the prototype dynamically adapts the metadata edition formulary to properties of the currently edited description. For instance, if the property "modifiable" of an input is checked to "yes", than new elements are displayed on the interface to describe the modification of the input.

Concerning the querying of descriptions, the application implements an important principle from the Semantic Web. To search a resource, the user does not necessarily have to use the exact terms involved in the metadata database. He may express a query with concepts and terms specific to his own context. For instance, he can specify the OS he uses on his machine, he can specify the format and structure of data he wants to apply a function to, he may also specify his expertness. Such a user query is interpreted by an inference engine to find if items in the metadata database matches it. This is supported by the use of ontologies to value some elements of the model. Some ontologies are mere classification. Others are equipped with axioms that reveal useful in matching a user query against the metadata database. The OWL language has been used to define these ontologies as well as the inference engine Sesame [W3C 04].

Concerning the browsing of descriptions, our system may adapt the presentation of metadata to the user who is browsing or querying them. This has been implemented for the grounding entity.
The system first computes the gap between the user context and the conditions of use of the resource, for instance
GAP= (UserContext.InputObjectsType==BDTopo.SurfaceHydroType) AND (Resource.profile.input1.logicalType==PolygonePlat).
It then relies on specific expert rules to compute the necessary steps to repair the GAP like
IF GAP=(…)AND(…) THEN REPAIR.STEP1=("apply SurfaceHydroType2PolygonePlat(SurfaceHydroType sht) to UserContext.Input").

This mechanism yields a grounding best fitted to the current user context.

CONCLUSION

This paper describes our proposal. The first part of it is a metadata model to describe "data manipulation resources" within IGN. These resources are abstract functions, implemented functions and software resources that encapsulate implemented functions. The second part of it is a catalogue application that contains expert rules to map a user query to the metadata database and expert rules to adapt some metadata to the user context.

An important perspective of this work is that it will allow us to assess the feasibility of sharing a function before possibly implementing Web Services to diffuse it to end users. Indeed, it is not enough to provide a XML interface to invocate methods that will realize a service, it is also necessary to provide enough knowledge to the user for him to correctly parameterize the Web Service and correctly interpret the returned value.

BIBLIOGRAPHY

Abd-el-Kader Y., Cataloguing Geographical Data Processing Tools, Conception and Exploitation of a Metadata Model, in proceedings of the International Cartographic Conference, Coruna, Spain, 2005

ISO, ISO/DIS 19119 : Geographic information – Services, Draft International Standard, 2001

Lutz M. Ontology-Based Service Discovery in Spatial Data Infrastructures. Workshop on Geographic Information Retrieval (GIR 2005), Bremen, Germany, 2005

OWL Services Coalition, OWL-S : Semantic Markup for Web Services, 2004

W3C, Web Services Description Language (WSDL), W3C Note, 2001

W3C, OWL Web Ontology Language, W3C Recommendation, 2004