INFORMATICA, 2009, Vol. 20, No. 3, 397–416

Dynamic Generation of Web Services for Data Retrieval Using Ontology

Ivan MAGDALENIC, Danijel RADOSEVIC
Faculty of Organization and Informatics, University of Zagreb
Pavlinska 2, HR-42000 Varazdin, Croatia
e-mail: ivan.magdalenic@foi.hr

Zoran SKOCIR
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia

Received: September 2007; accepted: July 2009

Abstract. Semantic Web is envisioned as semantic description of data and services enabling unambiguous computerized interpretation. Thanks to semantic description, computers can perform demanding tasks such as automation of discovery and access to heterogeneous data sources. Although this is possible with the existing technologies, combination of web services technology, ontologies and generative programming methods makes this simpler and more efficient. This paper presents the model for dynamic generation of web services for data retrieval from heterogeneous data sources using ontologies. Emphasis is on dynamic generation of web services customized to a particular user based on the request defined by ontology. The paper also describes a prototype of the model implementation. Some advantages of our approach over other approaches are also provided.

Keywords: web services, data retrieval, ontology, generative programming.

1. Introduction and Motivations

The original Web can be described as a range of documents and links between the documents where the publishers create the content and the people browse for information. Web 2.0 can be called the Service Web because it involves a new approach where the interaction, user creations and shaping of content become the most important issues. The next step is a Semantic Web, where semantic data description allows computers to discover and retrieve information. This paper presents how semantics can play an active role in data retrieval by following the architecture described in Magdalenic *et al.* (2006). Let us consider the following scenario: a user needs an application that automatically discovers web services and uses them for data retrieval. Until now, this scenario required the following steps: search of UDDI registers or Internet for web services, reading of a web service description and metadata in order to familiarize with their functionality, creation of client code for every chosen service and integration into the existing application. The problems immediately identified are finding the appropriate web service which returns

the desired data, and use of that particular web service. Both problems are addressed to the user instantaneously and the user must know how to solve them. It is important to know that the user is not interested in web services; he seeks data that can be obtained by them. The difference between a service and a web service is well-known and described in Baida *et al.* (2004). Technological solutions must meet the user's needs without the assumption that the user has profound knowledge of the applied technologies. Identification of the appropriate web service can be facilitated by adding semantic annotations to web service description. We are introducing new semantic annotations joined to the existing description of the web service and adopted for data retrieval from heterogeneous data sources. The same semantic annotations are being used for web service description and for description of the user's request. We believe that the use of web service technology can be improved and made easier if a service interface is customized to a particular user. Currently, user must create client code for every service because it is hard to expect equal interfaces for services of same functionality. If a service interface can be customized to a particular user then only the location of a new service must be added to the client's application. We propose the use of generative programming to solve service interface customization. We are also introducing a model for creation of web services for data retrieval from heterogeneous data sources, and for customization of a web service interface to the user's request defined by ontology. For model realization we used the web service technology, ontology and generative programming methods. The paper is organized as follows: related work is presented in Section 2. The model for web service creation and customization to the user's request is presented in Sections 3–10. Sections 11 and 12 describe implementation of the model presented in Sections 3–10. The conclusion is given in Section 13.

2. Related Work

This paper draws upon topics found in several research areas. The first research area refers to data retrieval from heterogeneous data sources; the second one refers to semantic description of the web service; the third one refers to interoperability and web services orchestration; finally, there is the research area of generative programming. A significant project in the area of data retrieval with web services is OGSA DAI. This is a middleware product that allows access to heterogeneous data sources, such as relation or XML databases using the web services technology. OGSA DAI web services allow execution of the queries upon data as well as retrieval of information about the services themselves. Its infrastructure consists of the server's and the client's part. The messages have the XML structure and use the web service technology for communication (Karasavvas *et al.*, 2005). The important advantage of this approach is a unique way of data management of different kinds of sources. The sessions are supported, thus extending the core web services functionality (Atkinson *et al.*, 2005). Having a client part written in Java is a restriction which brings many advantages and additional functionality, yet it gets along with basic principles of platform and language independence of web services. OGSA-DAI uses metadata to describe the web services. The web services must be semantically

described in order to ensure unambiguous interpretation of their contents, which is indispensable for the automated binding. The problem with semantic description of web services here is whether the basic program agents are able to locate, identify and combine these services. If this is done efficiently, dynamic binding of web services during runtime is of higher importance. Many researchers believe that the vision of a semantic web, which provides computers with unambiguous interpretation of the web content, addresses and solves this problem (Wang *et al.*, 2004; Cardoso, 2007). Ontologies are widely accepted state-of-the-art knowledge representations and have, thus, been identified as the central enabling technology for the Semantic Web. Their extensive usage allows semantically enhanced information processing as well as interoperability support (Roman *et al.*, 2007). Ontologies can assist in communication between humans, achieve interoperability among software systems and improve design and quality of software systems (Cardoso, 2007). Lately, ontologies have become the focus of research in several other areas, including knowledge engineering and management, information retrieval and integration, agent systems, the Semantic Web, and e-commerce. Availability of formal ontologies is crucial for the success of the Semantic Web (Benslimane *et al.*, 2007). There are several researches aiming to create ontology of a specific domain, e.g., the business process domain (Ciuksys and Caplinskas, 2007). In the area of semantic enrichment of the web service description, the most significant projects are OWL-S (Martin *et al.*, 2007), WSMO (Roman *et al.*, 2007) and METOR-S (Sheth *et al.*, 2007). OWL-S describes the architecture and the markup language for supporting web services and the Semantic Web. Web services provide business logic and the Semantic Web provides unambiguous and understandable content. OWL-S tries to express web services in the language understandable to computers in order to enable automation of various aspects, such as discovery, invocation and composition. Semantic description of web services comes in separate files and is covered by three classes: Service Profile describes the service and gives information about its content; Service Model describes the service process decomposed into sub processes and Service Grounding describes concrete details of how to invoke the service. OWL-S offers a good way to express semantics of web services, which can be of a great benefit to understanding the web service. A good semantic description achieved by the OWL-S standards has taken significant effort. We believe that some domains, like data retrieval from the heterogeneous data sources, can be covered by a simpler method, and thus by a more efficient, semantic description. In addition to specification, WSMO, unlike OWL-S, offers supporting tools. In order to describe ontology and the web services, it defines the language called the Web Service Modelling Language (WSMO) (Scicluna, 2007). In the domain of data retrieval it shows the same advantages and disadvantages as OWL-S. As a part of METEOR-S project, a framework for describing semantic annotations in WSDL definitions of the web services (Patil *et al.*, 2004) has been developed. In the evolutionary approach to the existing WSDL specification, semantic annotations are added to give semantic meaning (Sivashanmugam *et al.*, 2003). A great advantage of this approach is reliance on the existing specification, which is beneficial for wide acceptance of this standard. It is the best solution for the enclosed domain where annotations are sufficient to give the whole semantic picture of the web service. This approach is also used in our

work. There are two conditions that must be satisfied in order to achieve efficient web services orchestration. The first one addresses functionality and the second addresses interfaces. The former one is solved in the context of ontology and the Semantic Web (Sattanathan *et al.*, 2006). The latter can be solved by developing the framework and simple language as a middleware between the services (Skrobo *et al.*, 2005). Orchestration of web services is much simpler if the same data is reached through different interfaces. User can then choose the interface most appropriate to his/her needs and expedite the development of a distributed application. It is even better if the web service interface is accommodated to special requirements of each end user. This can be performed only for a specific area, e.g., access to data. Generative programming is a discipline of automatic programming which started, under that name, in the late 90's. According to the definition, generative programming represents "... designing and implementing software modules which can be combined to generate specialized and highly optimized systems fulfilling specific requirements" (Eisenecker, 1997). Aspiration to programming code optimization makes, according to Eisenecker (1997), the main specific difference to other techniques of automatic programming. The main goals of generative programming are, according to Czarnecki and Eisenecker (2000):

- to decrease the conceptual gap between program code and domain concepts (known as achieving high intentionality);
- to achieve high reusability and adaptability;
- to simplify managing many variants of a component; and
- to increase efficiency (both in space and execution time).

Generative programming was the result of aspiration to increase software-producing productivity by producing it in a way comparable with industrial production. Due to some weaknesses, observed by Guerraoui (1996) and Ousterhout (1998), current object-oriented programming is not able to fully fulfil these aspirations. For this reason, some new disciplines of programming were developed in the middle of 90's, hoping to succeed the object paradigm. One of these disciplines is the so-called Aspect Oriented Programming (AOP) (Guerraoui, 1996), which is one of the basic generative programming disciplines (along with metaprogramming, generic programming, object-oriented programming and domain engineering; according to Czarnecki (1999). Generative programming was originally considered mostly as a discipline of object-oriented programming. But, as announced by Sells (2001), there were some attempts to connect generative programming with scripting languages. Several design projects for specialized scripting languages aimed at generative programming also appeared. Two of them are Open Promol (Stuikys and Damasevicius, 2000) from Lithuania and Codeworker (Lemaire, 2007) from France.

3. The Model for Dynamic Generation of Web Services for Data Retrieval from the Heterogeneous Data Sources Using Ontologies

Fig. 1 describes the model for dynamic generation of web services for data retrieval from heterogeneous data sources and for interface adjustment to user's request described by ontology.

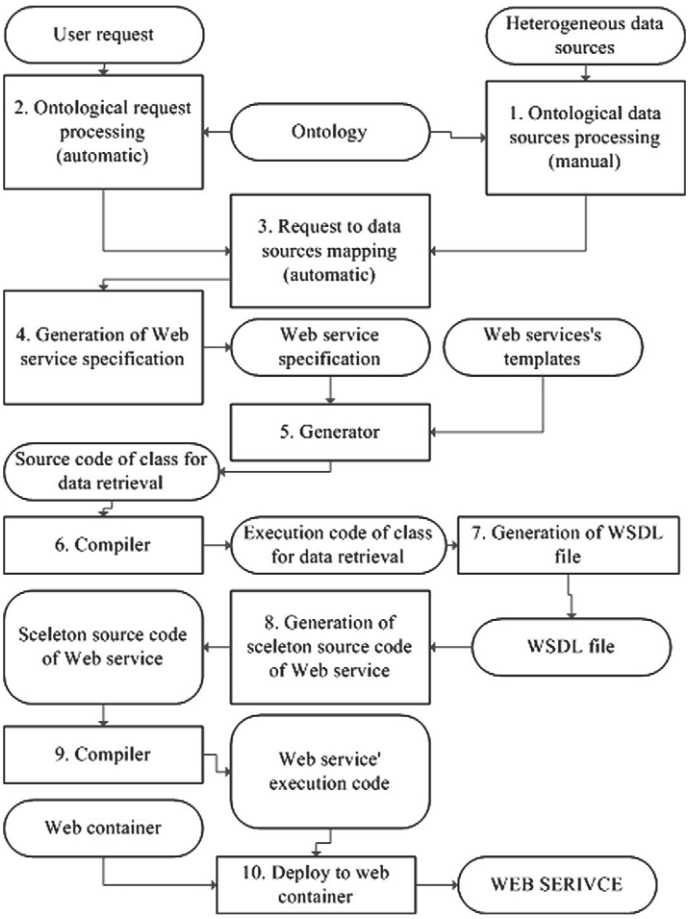

Fig. 1. The model for dynamic generation of web services for data retrieval from the heterogeneous data sources and for interface adjustment to user's request described by ontology.

Parts of the model and their functionality are described as well. The purpose of the model is to create programmable framework which will enable simple and efficient data retrieval by the web services technology. Special emphasis is on the usage by users unfamiliar with the web service technology. A short description of the model is given first, and a detailed functionality description given in later sections. Data sources are described in the first step. Basic information about data sources is stored in the internal database. Columns or nodes of data sources to ontology are mapped. In the second step the user's request is analyzed, which is actually the WSDL file enriched by semantic annotations. Next we have the user's request and data sources described by the common language-ontology. If the items from the user's request exist as items from registered data sources, an appropriate web service can be made (Step 3). Interface information is extracted from

the WSDL file. Interface information and a semantically defined request are the basis for generation of a web service specification (Step 4). The web service specification is used by a source code generator to generate web service source code (Step 5). The source code has to be compiled (Step 6) and it is the input for generation of the WSDL file (Step 7). The skeleton code of the web service is generated upon the WSDL file (Step 8) and then compiled (Step 9). The final step is a web service deployment to a web container (Step 10). URL of the web service WSDL file is returned to the user as a final outcome of the whole process.

4. Semantic Processing of the User's Request

Industrial standard for web service description is the Web Service Description Language, which has several releases. Although the latest release of specification is 2.0, our model is based upon a specification version 1.2 (Chinnici *et al.*, 2003) because it is supported by most applicable software. WSDL defines the model and XML the format for the web service description. Description of the web service consists of two parts: abstract description of functionality and concrete description of details as to where to find the service and how to invoke it. Service description in the WSDL format can be stored in UDDI registers so that the service can be located. UDDI search mechanisms are not good enough for automatic locating because they lack the semantics in service descriptions. Introduction of semantics into the web service description is the aim of several projects. A detailed comparison between them is given by Verma *et al.* (2003). The closest approach to our work is the one from the METEOR-S project where the Web Service Sematics model – WSDL-S is developed. The WSDL-S approach is evolutionary; it consists of adding the semantic annotation to the existing WSDL standard. The same approach is also supported in this work, all be it with some important differences. We believe that semantic annotations from the WSDL-S are not sufficient to support dynamic service generation for data retrieval. WSDL-S has the attribute named "modelReference" used for connection of the WSDL element with the external semantic element. That element cannot cover the instances where concepts of the external world are described by classes and where the concept attributes are described by properties. A concrete example is in the area of the relation database description. It is recommended that the relations (tables) are described by classes and the attributes of relations (columns) are described by properties (Peter Alesso and Craig, 2005). Thus we propose the new elements which will cover the concepts and the concept attributes and enable unambiguous connection of the WSDL elements with the external semantic elements. The external semantic elements refer to elements contained in a different domain ontologies, taxonomies or other semantic annotations. We use a primary OWL which supports the vocabulary of a concrete domain and the relation of equivalency. Relation of equivalency enables binding the same elements within different domain ontologies. The following namespace is introduced (Table 1):

Table 1
Namespace definition

Prefix	Namespace
dwsg	`http://www.foi.hr/xmlns/DynamicWebServiceGeneration/`

The following elements and attributes for semantic description are introduced:

- extension attribute **modelClass**, to handle one-to-one mapping of the schematic elements to the concepts in a semantic model;
- extension attribute **modelProperties**, to handle one-to-one mapping of the schematic elements to the concept properties in a semantic model;
- extension attribute **filterOperator**, to carry the comparison operator which will be used for data filtering. Accepted values are: equal, notEqual, greaterThen, greaterOrEqual, lessThen, lessOrEqual, contains, startsWith, endsWith;
- extension attribute **responseType**, to carry the information about a response format. Accepted values are: single – for response with single value, complexXml – for a response with more values packed in the XML format;
- new element **response**, specified as a child element of the message element. A response element is used if the response must have two or more elements packed in the XML format;
- attribut name (extension attribute **name**, to carry the information for name of the responded XML element); and
- extension attribute **dataSourceChoice**, to carry the information which will be used in making the source choice. Accepted values are: new – for selecting the latest source (the last registered source), all – for selecting all acceptable sources in one service, separate – for creating more services (one service per acceptable data source), optimal – choice of selection of only one data source is left to the server's internal logic.

XSD definition of the introduced extensions is shown in Fig. 2.

Fig. 3 shows a part of the WSDL file with examples of introduced extension elements and attributes.

The example given in Fig. 3 shows the semantically described all important parts of the WSDL file which give unambiguous information for data exchange to computers. Because the application domain is data retrieval from heterogeneous data sources, semantic description is made on the input and output messages. The input message is carrying information about the filtered data and filtering operation. The information about the semantic is carried by the attributes *dwsg:modelClass* and *dwsg:modelProperty*, and filter operation is carried by the attribute *dwsg:filterOperator*. Data type can be extracted from the WSDL attribute type. The same attributes are used to describe the output message with the addition of the element *dwsg:response*, used if a response contains more then one piece of data. In that case the return value is the XML element which consists of sub-elements, each semantically described by the *dwsg:response element*.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.foi.hr/xmlns/
DynamicWebServiceGeneration/"
 xmlns:dwsg=" http://www.foi.hr/xmlns/DynamicWebServiceGeneration/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
<attribute name="modelClass" type="anyURI" use="optional"/>
<attribute name="modelProperties" type="anyURI" use="optional"/>
<attribute name="filterOperator" type="anyURI" use="optional"/>
<attribute name="responseType" type="anyURI" use="optional"/>
<attribute name="dataSourceChoice" type="anyURI" use="optional"/>
<element name="response">
 <complexType>
  <complexContent>
   <extension base="wsdl:documented">
    <attribute name="name" type="anyURI" use="optional"/>
    <attribute name="modelClass" type="anyURI" use="optional"/>
    <attribute name="modelProperties" type="anyURI" use="optional"/>
   </extension>
  </complexContent>
 </complexType>
</element>
</schema>
```

Fig. 2. DWSG scheme.

```
<wsdl:message name="m1Request">
<wsdl:part name="in0" type="xsd:float"
dwsg:filterOperator="lessOrEqual" dwsg:modelProperty="http://
www.foi.hr/dynamicweb/Yahoo.owl#priceCash"
dwsg:modelClass="http://www.foi.hr/dynamicweb/
Yahoo.owl#Computers">
</wsdl:part>
</wsdl:message>
<wsdl:message name="m1Response">
<wsdl:part name="m1Return" type="soapenc:string"
dwsg:responseType="complexXml">
<dwsg:response dwsg:modelClass="http://www.foi.hr/dynamicweb/
Yahoo.owl#Computers" dwsg:modelProperty="http://www.foi.hr/
dynamicweb/Yahoo.owl#code" dwsg:name="elementName"/>
<dwsg:response dwsg:modelClass="http://www.foi.hr/dynamicweb/
Yahoo.owl#Computers" dwsg:modelProperty="http://www.foi.hr/
dynamicweb/Yahoo.owl#priceCash" dwsg:name="elementName"/>
</wsdl:part>
</wsdl:message>
<wsdl:portType name="AllSources" dwsg:dataSourceChoice="all">
wsdl:operation name="m1" parameterOrder="in0">
<wsdl:input name="m1Request" message="impl:m1Request">
</wsdl:input>
<wsdl:output name="m1Response" message="impl:m1Response">
</wsdl:output>
</wsdl:operation>
</wsdl:portType>
```

Fig. 3. An example of the semantic annotations in the WSDL file.

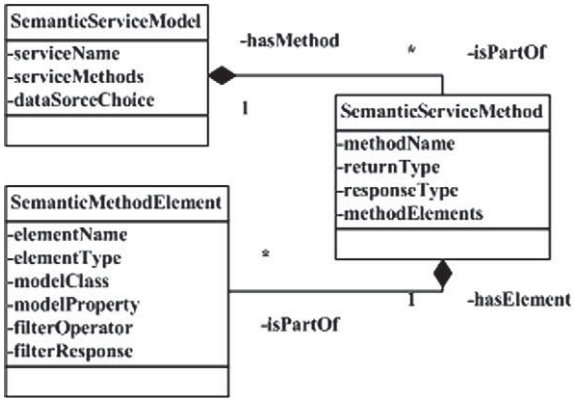

Fig. 4. The UML class diagram of request defined by ontology.

The service interface enriched by the semantic annotations must be parsed, after which a semantic model of the request is created. A semantic request model described by the UML classes is shown in Fig. 4. A semantic model consists of three classes. *SemanticServiceModel* contains the data about a service name and has a list of methods. *SemanticServiceMethod* gives the information about a method name, return type, response type as described by the attribute dwsg:responseType and has a list of elements. *SemanticMethodElement* contains the information about a method attributes and information about the response data.

5. Semantic Processing of Data Sources

When the request is semantically described, data sources also must be described using the same language in order to enable the search for the required data. In the past few years several languages for semantic description have been developed and some are still under development (e.g., RDF, OIL, DAML + OIL, OWL etc.). The Web Ontology Language (OWL) is the latest markup language for writing ontology and is our first choice for our model implementation. The area of mapping data sources to ontology is still the focus of research and literature covers it using various approaches. The main problem is that the domain covered by the table is not always the same as the domain covered by the ontology concept. The domains can overlap only partially. The problem with automated mapping of data sources to ontology is that browsing the content is often insufficient to show what is actually represented. For example, the columns containing numbers can represent column identification, cash price, price for payment with credit cards, discount, etc. There are some solutions for mapping relational databases to ontology, as described in Barrasa *et al.* (2004). There is also the approach of building personalized ontologies by analyzing HTML pages generated from data from relational databases (Benslimane *et al.*, 2007). Because there is no uniform solution for all data sources we recommend manual mapping of data sources to ontology. For that purpose we have developed the web interface.

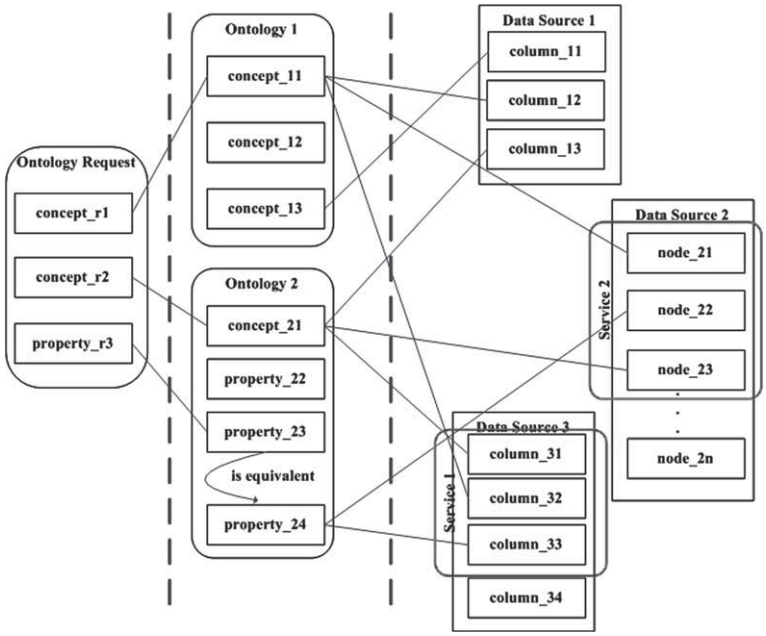

Fig. 5. The search for data sources by ontology.

6. Mapping User Requests for Data Sources

When the user's request and data sources are described by ontology, the search can be made through data sources in order to decide which best suits the user's request. Fig. 5 describes one of such search cases. The principle of search is as follows:

1. Every data source is being searched to see if it contains all items from the user's request. The equivalent classes and properties must be considered.
2. If a data source has every item from the user's request (data sources 2 and 3, Fig. 5) the internal database is queried for the given service.

If the service already exists, its URL is returned to the user. If the service does not exist, one or more services will be generated, depending on the value of the attribute *dwsg:dataSourceChoice*.

7. Source Code Generation

A source code generator is described by the Scripting model guidelines, introduced by Radosevic (2006). The Scripting model was originally developed for generative programming based on the scripting languages (Radosevic, 2005). Unlike the object model, it is oriented towards defining specified aspects of future applications within a specified problem domain, rather than on all application functionalities, as these are defined on the lower level, in the program code templates (within the scripting model these are called

the metascripts). The aspects, according to Kiczales *et al.* (1997), represent the features not strictly connected to organizational units of an individual program, like functions or classes; therefore, they can appear within different application parts. Consequently, the crosscutting concerns that cannot be expressed by entities can be abstracted and encapsulated in the aspects (Yao *et al.*, 2005). However, a connection model is needed to implement the aspects. Principally, the offered model is a type of a join point model (Kandé *et al.*, 2002). Join points are well-defined points during program execution, around which extra aspectual code can be executed (Pengcheng and Lieberherr, 2005). Furthermore, a scripting model is not based on types, i.e., it is a type-free system (Albano *et al.*, 1989), because the connecting points in a scripting model do not represent classes and their objects, only connections (they have their names only, but not the property) between the metaprograms and the properties defined in the application specification (Radosevic, 2005). The fact that the connection points do not have properties makes graphical representation of a join points model easier. A scripting model consists of two graphical diagrams (or of the equivalent textual specifications). Consequently, it is simpler than the models based on UML (Radosevic, 2005). The first diagram is called the *specification diagram*. It defines the structure of the application specification within the generation system (Fig. 7). The generation system generates the application within its problem domain, which is designated by the program code templates (metascripts). The rules for connecting metascripts to application specification are defined in the second diagram – the *metascripts diagram* (Fig. 10); (Radosevic, 2005).

8. Steps for Building a Source Code Generator

The following steps are necessary for building a source code generator:

- detailed analysis of the specific problems which appear by data retrieval;
- building of the web services prototypes for data retrieval;
- building of the source code generator by the following steps:
  1) analysis of web services prototypes,
  2) detachment of common aspects,
  3) separation of the concerns according to the guidelines of aspect-oriented programming (AOP),
  4) design of the source code templates and of the specification elements.

A web service generator generates a source code using the application specification and appropriate code templates as shown in Fig. 6. A generated source code is compiled and made ready to expose as a web service.

9. Generator Specification

Web services generator specification has four levels, as shown in Fig. 7.

Marks *package* and *serviceName* refer to the generated application as a whole (all web services contain these elements). Some methods have their specific properties, defined as the repeated sequences of marks under the mark *methodName*. The mark *returnType* gives the information about return type and is specified on the level of a particular

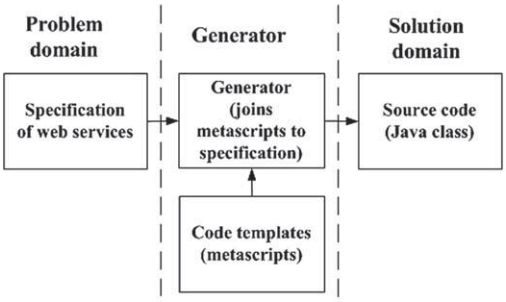

Fig. 6. Generator of web services.

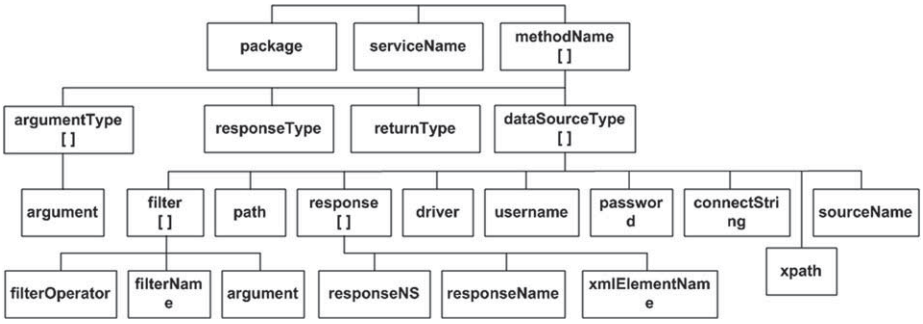

Fig. 7. The specification diagram.

method. Method's *arguments* are specified under the mark *argumentType*. Each argument has its own type (argumentType) and value specified by the mark argument. The mark *responseType* gives the information about a response type, e.g., if it is about a simple or a complex response. Data sources are defined by the mark *dataSourceType* under which there are relevant information for the access to a data source defined by: *path, driver, username, password, connectString, sourceName and xpath*. Data filtration is defined by marks *filter, filterOperator, filterName and argument*. Mark *argument* is here used for binding the input argument of methods with a particular column or a node of data source. Return data are defined by marks *responseName*, *responseNS* and the name of a node in XML format defined by *xmlElementName*. An example of a specification according to the diagram from Fig. 7 is shown in Fig. 8. The specification from Fig. 8 specifies a class with one method which fetches the values from two different data sources. The first data source is the XML file and the second data source is a table from the relational database. A generated class integrates the data from two data sources and returns them to the user.

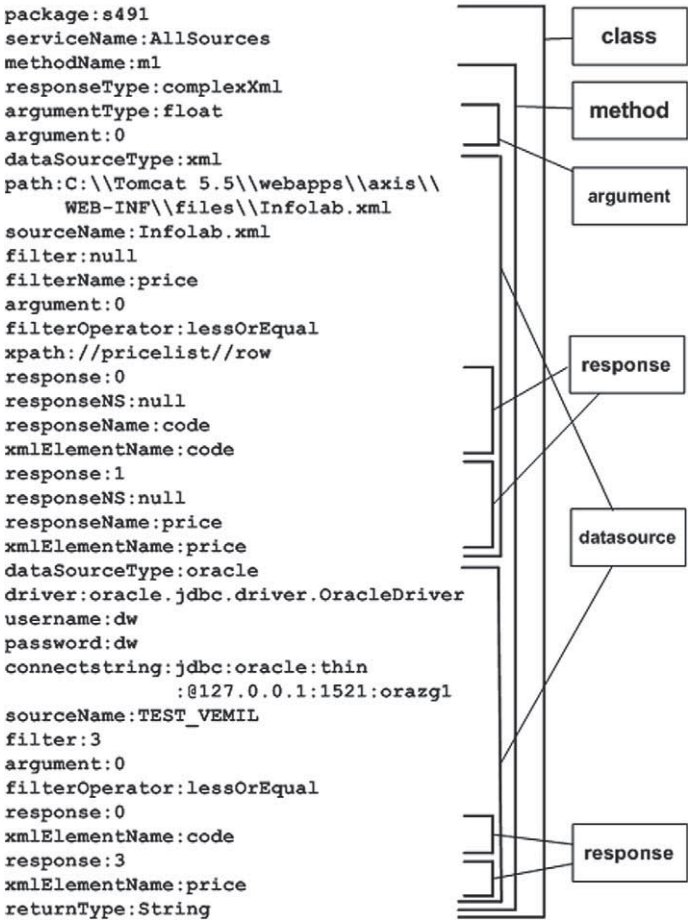

Fig. 8. An example of the specification for source code generator.

10. Metascripts Diagram

A metascripts diagram defines binding of the source code templates to the properties defined in the application specification. Fig. 10 shows a metascript diagram of the web services generator which defines the relationship between the metacsripts and application. Basic elements used in the diagram are shown in Fig. 9.

A metascript is a template used in generation of its implementation – program code in the target programming language. It is represented by a rectangle. Metascripts contain replacing tags – tags replaced by designated data or a program code. The tags are designated within the metascripts diagram by the link element. A link connects a metascript to a replacing source and (optionally) to one or more metascripts of the lower level that are used to generate a replacing code. It is represented by a triangle, with one of the angles pointing downwards, containing the name of the replacing tag. A code can be replaced in

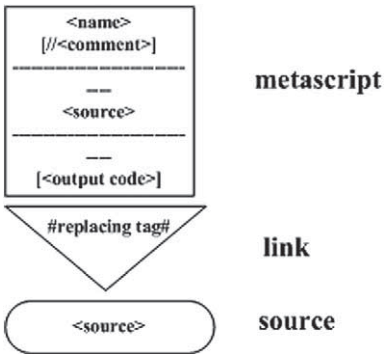

Fig. 9. Elements of the metascripts diagram.

one of the following two ways:

- through direct replacement of a tag by the source data (if there are no metascripts of the lower level) and
- by replacement of a tag by a metascript of the lower level (which also contains its own links and sources).

A source represents a particular parameter defined by an appropriate tag in a diagram of the application specification parameters. Sources can be defined as group parameters (having a tree-like structure), in which case their further use must be defined on lower levels of the metascripts diagram (by specifying particular sources). Let us return to Fig. 10. The metascript on the first level is called CLASS and represents a template of the whole class to be generated. The links #*serviceName*#, #*package*# i # *sourceId*# are directly replaced by appropriate values from the specification. The metascript METHOD is on the second level and represents the template of a particular method. The number of method arguments is gratuitous and replacement of the link #*arguments*# is defined on the next level by usage of the metascript ARGUMENTS. Some parts of these methods depend on the type of data source. Implementations of access to different types of sources are settled to the metascripts SOURCES. The link to metascript sources is *dataSourceType*. Some other details depend on the specification and the type of data sources. Finally, there are five levels of templates.

11. Example of a Source Code Generation

An example of a generated source code is presented in Fig. 11. Source code is generated according to the specification from Fig. 8. Fig. 11 shows only a part of the whole class, specifically access to data source through the XML type. The generated source code is JAVA class named SourceXML.java in the package *service/s491/*. Generated JAVA class contains the method *m1* which has one argument of the type *float* and returns the value of the type *String*.

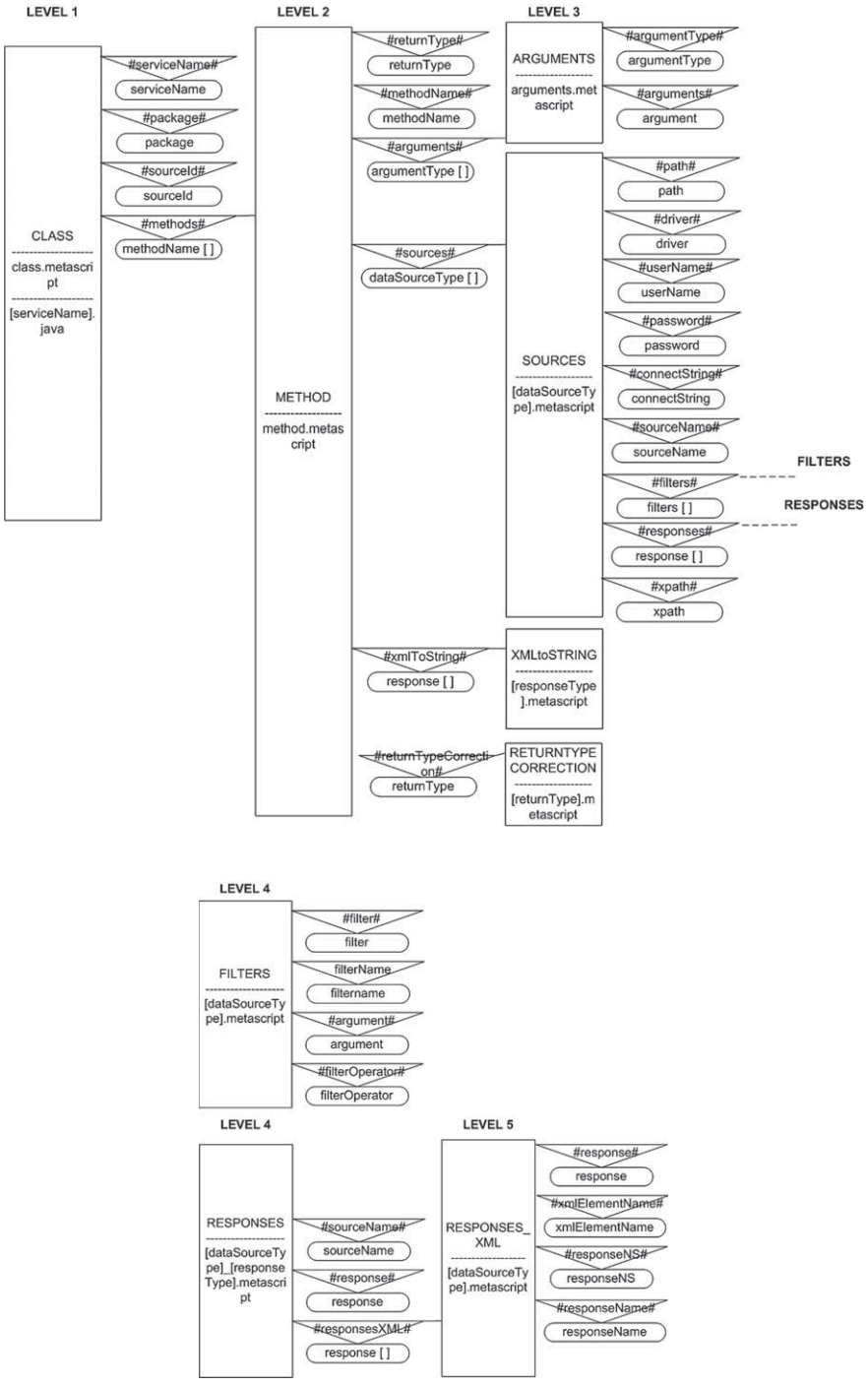

Fig. 10. The metascripts diagram.

Fig. 11. Generated Java class.

Finally, some properties from the specification end up in the generated source code directly. Some specification properties implicate the rules of binding the specification to appropriate templates of the source code, as defined in the metascripts diagram (Fig. 10).

12. Model Prototype

Prototype of the model was built using the programming language JAVA. We used the following third party products: Tomcat 5.5 as web container, Axis 1.3 as software for generation of web service's skeleton code and for creation of WSDL files, Jena 2.4 for managing WSDL files. Web interface for our implementation was built using JSP technology and Oracle RDBMS was a database for storage of all internal data. The web interface

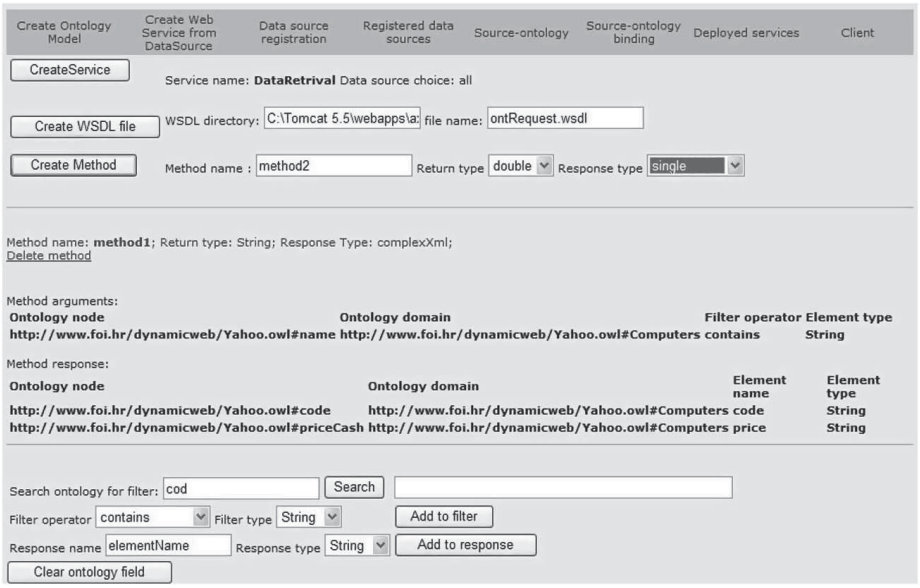

Fig. 12. Web interface for web service generation.

was used for data source registration, for ontology registration and for mapping of data sources to ontology. We also built a web interface where data source owners can create web services for data retrieval in a simple fashion. An example of the web interface is presented in Fig. 12.

The main part of the prototype is the web service named *dynamicws* which can be invoked by users and which reads semantically described WSDL files as input arguments. When such WSDL file is read, ontological model of a request is created. *Dynamicws* performs one of the following: it returns the URL from the existing web service; it creates one or more new web services and returns their URLs; it prevents the service from returning anything if a user request cannot be fulfilled. It is sufficient to know the URL of the *dynamicws* web service in order to query data. Client web interface is developed for testing purposes. A client can generate his/her client code based on the URLs returned from the *dynamicws* web service, following which the service can be invoked to retrieve the data.

13. Conclusions

This paper presents the model for dynamic generation of web service for data retrieval from heterogeneous data sources using ontology. Binding of the web services technology, ontology and methods of generative programming enables the development of a modular system for creation of web services customized to a particular user. Application of ontology in semantic description of the web services enables computers to unambiguously

interpret the user's request, allowing either the creation of new web services or recognition of the existing web services. There are many advantages of the ontology-supported web services for data retrieval compared to the traditional approach which includes programming access to data sources or compared to web services which do not support ontology. The most obvious are automated implementation of new web services, rapid development, and creation of the services customized to user's request where created client code can be used to access more data sources. In addition, the web services technology increases the safety level by allowing access only to data sources registered by the data owner. When the user has an application which data input are described in a way described by our model, discovery and invocation of new data sources could be automated. In that case the end user does not need to have profound knowledge of complex technical background. Our future work will focus on the automated mapping of data sources to the existing ontology, which will add functionality to our model.

References

Albano, A. *et al.* (1989). A framework for comparing type systems for database programming languages. In: *Computer Science at the University of St Andrews.* St Andrews, UK. Available at: `http://www.cs.st-andrews.ac.uk/files/publications/download/ADG+89.pdf`.

Atkinson, M. *et al.* (2005). A new architecture for OGSA-DAI. In: *Proceedings of the UK e-Science All Hands Meeting 2005*. Available at `http://www.allhands.org.uk/2005/proceedings/papers/432.pdf`.

Baida, Z., Gordijn, J., Omelayenko, B., Akkermans, H. (2004). A shared service terminology for online service provisioning. In: *ICEC '04*: *Proceedings of the 6th International Conference on Electronic Commerce*, pp. 1–10.

Barrasa, J., Corcho, O., Gomez-Perez, A. (2004). *R2O, An Extensible and Semantically Based Database-to-Ontology MApping Language*. Ontology Engineering Group, Departmento de Inteligencia Artificial, Facultad de Informatica, Universidad Politecnica de Madrid, Espana.

Benslimane, S.M., Rahmouni, M.K., Benslimane, D. (2007). Extracting personalised ontology from data-intensive web application: An HTML forms-based reverse engineering approach. *Informatica*, 18(4), 511–534.

Cardoso, J. (2007). *Semantic Web Services*: *Theory, Tools, and Applications*. Information Science Reference. Hershey.

Chinnici, C., Gudgin, M., Moreau, J., Weerawarana, S. (2003). *Web Services Description Language* (*WSDL*). Version 1.2. W3C Working Draft. Available at: `http://www.w3.org/TR/2003/WD-wsdl12-20030124/`.

Ciuksys, D., Caplinskas, A. (2007). Reusing ontological knowledge about business processes in IS engineering: Process configuration problem. *Informatica*, 18(4), 585–602.

Czarnecki, K., Eisenecker, U. (2000). *Generative Programming*: *Methods, Tools and Applications*. Addison-Wesley.

Czarnecki, K. (1999). *Generative Programming and GMCL.* Technische Universität Ilmenau, Fakultät für Informatik und Automatisierung.

Eisenecker, U. (1997). Generative programming (GP) with C++. In: *Proceedings of Modular Programming Languages*. Springer-Verlag, Heidelberg/Linz.

Guerraoui, R. (1996). *Strategic Directions in Object-Oriented Programming*. ACM Computing Surveys. Baltimore.

Kandé, M.M., Kienzle, J., Strohmeier, A. (2002). From AOP to UML – A bottom-up approach. In: *1st International Conference on Aspect-Oriented Software Development*, Enschede, The Netherlands.

Karasavvas, K. *et al.* (2005). Introduction to OGSA-DAI services. In: *Scientific Applications of Grid Computing*. In: *Lecture Notes in Computer Science*, Vol. 3458/2005. Springer, Berlin/Heidelberg.

Kiczales, G. *et al.* (1997). Aspect-oriented programming. In: *Proceedings of the European Conference on Object-Oriented Programming*. *LNCS*, Vol. 1241. Springer-Verlag.

Lemaire, C. (2007). *CODEWORKER Parsing Tool and Code Generator*, *User's Guide & Reference Manual*. Available at `http://codeworker.free.fr/CodeWorker.pdf`.

Magdalenic, I., Vrdoljak, B., Skocir, Z. (2006). Towards dynamic web service generation on demand. In: *Proceedings of the International Conference on Software, Telecommunications and Computer Networks 2006*, Split, Croatia.

Martin, D. *et al.* (2007). *OWL-S: Semantic Markup for Web Services*. `http://www.w3.org/Submission/OWL-S/`.

Ousterhout, J.K. (1998). Scripting: Higher level programming for the 21st century. *IEEE Computer Magazine*, March.

Patil, A., Oundhakar, S., Sheth, A., Verma, K. (2004). METEOR-S web service annotation framework. In: *The Proceedings of the Thirteenth International World Wide Web Conference*, New York, pp. 553–562.

Pengcheng, W., Lieberherr, K. (2005). Shadow programming: Reasoning about programs using lexical join point information. In: *Generative Programming and Component Engineering Conference* (*GPCE*) *2005*. In: *Lecture Notes in Computer Science*, Vol. 3676/2005. Springer, Berlin/Heidelberg, pp. 141–156.

Peter Alesso, H., Craig, F.S. (2005). *Developing Semantic Web Services*. A.K. Peters, Ltd.

Radosevic, D., Klicek, B., Dobsa, J. (2006). Generative development using scripting model of application generator. In: *DAAAM International Scientific Book 2006*, Vienna, Austria.

Radosevic, D. (2005). Integration of generative programming and scripting languages (doctoral thesis). In: *DAAAM International Scientific Book 2006*. Fakultet organizacije i informatike, Varazdin, Croatia.

Roman, D., Lausen, H., Keller, U. (Eds.) (2007). *Web Service Modeling Ontology*. `http://www.wsmo.org/TR/d2/v1.2/`.

Sattanathan, S., Narendra, N.C., Maamar, Z. (2006). Ontologies for specifying and reconciling contexts of web services. *Electronic Notes in Theoretical Computer Science*, 146(1), 43–57.

Scicluna J, (2007). *Web Service Modeling Language*. `http://www.wsmo.org/wsml/`.

Sells, C. (2001). Generative programming: Modern techniques to automate repetitive programming tasks. *MSDN Magazine.* `http://msdn.microsoft.com/msdnmag/issues/01/12/GenProg/GenProg.asp`.

Sheth A., Miller, J., Budak Arpinar, I., Verma, K. (2007). *METEOR-S: Semantic Web Services and Processes*. `http://lsdis.cs.uga.edu/projects/meteor-s/`.

Sivashanmugam, K., Verma, K., Sheth, A.P., Miller, J.A. (2003). Adding semantics to web services standards. In: *Proceedings of the 1st International Conference on Web Services*, Las Vegas, Nevada, pp. 395–401.

Skrobo, D., Milanovic, A., Srbljic, S. (2005). Distributed program interpretation in service-oriented architectures. In: *The 9th World Multi-Conference on Systemics, Cybernetics and Informatics*, Orlando.

Stuikys, Damasevicius (2000). Scripting language open PROMOL and its processor, *Informatica*, 11(1), 71–86.

Verma, K., Gudgin, M., Moreau, J., Weerawarana, S. (2003). *Configuration and Adaptation of Sematic Web Processes*. PhD dissertation. The University of Georgia.

Wang, H., Huang, J.Z., Qu, Y., Xie, J. (2004). Web services: Problems and future directions. *Journal of Web Semantics*: *Science, Services and Agents on the World Wide Web*, 1(3), 309–320.

Yao, Z., Zheng, Q., Chen, G. (2005). AOP++: A generic aspect oriented programming framework in C++. In: *Generative Programming and Component Engineering Conference* (*GPCE*) *2005*. In: *Lecture Notes in Computer Science*, Vol. 3676/2005. Springer, Berlin/Heidelberg, pp. 94–108.

I. Magdalenic is a PhD student and research associate at University of Zagreb, Faculty of Organization and Informatics, Varazdin, Croatia. His main research interests include development and application of new web technologies and e-commerce.

D. Radosevic is assistant professor at University of Zagreb, Faculty of Organization and Informatics, Varazdin, Croatia. His main research areas are generative programming and textmining.

Z. Skocir is a professor at University of Zagreb, Faculty of electrical engineering and computing, Zagreb, Croatia. The main research interest are e-commerce, databases and data warehousing.

Duomenims ieškoti panaudojant ontologijas skirtų pasaulinio tinklo paslaugų dinaminis generavimas

Ivan MAGDALENIC, Danijel RADOSEVIC, Zoran SKOCIR

Pasaulinis semantinis tinklas šiandien yra suvokiamas kaip jame saugomų duomenų ir jo teikiamų paslaugų toks semantinis aprašas, kurį vienareikšmiškai gali interpretuoti kompiuteris. Pasinaudodamas tokiais aprašais, kompiuteris gali vykdyti nurodytas užduotis, pavyzdžiui, automatiškai surasti reikiamų heterogeninių duomenų šaltinius ir organizuoti prieigą prie tų duomenų. Nors šitą galima padaryti esamomis pasaulinio tinklo paslaugų teikimo technologijomis, tačiau, papildžius jas ontologijomis ir generatyviojo programavimo metodais, tai vyksta paprasčiau ir daug efektyviau. Straipsnyje aprašyta, kaip, panaudojant ontologijas, dinamiškai generuoti pasaulinio tinklo paslaugas, atliekančias paiešką heterogeniniuose duomenų šaltiniuose. Pagrindinis dėmesys skirtas klausimui, kaip, panaudojant ontologiją konkretaus vartotojo pateiktoms užduotims analizuoti, dinamiškai generuoti paslaugas, kurių interfeisai būtų pritaikyti tam vartotojui. Straipsnyje taip pat yra aprašyta bandomoji pasiūlyto metodo realizacija ir aptarti svarbiausieji to metodo pranašumai prieš kitus panašios paskirties metodus.