STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

XRDL: A VALID DESCRIPTION LANGUAGE FOR XML-RPC

DIANA TROANCĂ[1] AND FLORIAN BOIAN[1]

Abstract. XML-RPC standard does not define any Description Language for this web service. XRDL started as an open source project with the purpose of being XML-RPC Description Language. The open source project already includes some applications for XRDL (automatic XRDL generation for servers and clients written in PHP or C++/Qt [3]), but it has not yet been demonstrated that XRDL can be used as a Description Language without any exceptions. This paper intends to prove that XRDL is a valid XML-RPC Description Language, so the focus can move back on its applicability and further development.

1. Introduction

The purpose of this paper is to demonstrate that XRDL is a valid Description Language for XML-RPC services. This includes the fact that any XML-RPC service can be described by a XRDL document and that any web service described by a XRDL document is a valid XML-RPC service. Therefore the demonstration will consist of two parts.

XRDL [3] started as an open source project. It has not yet been accepted as a standard and it is not widely accepted as the XML-RPC description language [8]. Some researchers accept it as XML-RPC Description Language and others argue its utility. The open source project already implements some applications for XRDL, but they do not demonstrate the validity of XRDL as a Description Language for XML-RPC services.

Received by the editors: April 17, 2013.

2010 *Mathematics Subject Classification.* 03B48, 03F60.

1998 *CR Categories and Descriptors.* H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based services*;

Key words and phrases. XRDL, XML-RPC, Web Services.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

XRDL is defined by a XSD-schema that describes the structure of a XRDL valid document. As defined in the XSD-Schema it has two wide sections for describing the service:

- a section where complex types used by the service can be defined and
- a section that describes the methods that the service offers.

In the types section of a XRDL document we can define complex types used by the web service. The second section describes all the methods offered by the service with the parameters needed and the result that the method will return. The data type of the parameters and the result is specified using the "type" attribute. According to the definition of XRDL the value of this attribute is a string. In order to generate a valid XML document it can take any value that represents a valid data type in XML, but XML has a lot of built-in data types.

The specification of XML-RPC [4] defines six simple types and two compund types that can be defined as a collection of more simple or compound elements. The basic data types are presented in Table 1 [5, 8].

TABLE 1. XML-RPC Data Types

Type	Value	Example
int or i4	32-bit integers	<i4>35</i4> <int>2</int>
double	64-bit floating-point numbers	<double>-3.456</double>
boolean	0 (false) or 1 (true)	<boolean>0</boolean>
string	ASCII text or Unicode	<string>How are you</string>
dateTime.iso8601	Dates in the following format: CCYYMMDDTHH::MM:SS	<dateTime.iso8601>20130903T13:11:05 </dateTime.iso8601>
base64	Binary data encoded as base 64 (as defined in RFC 2045)	<base64>base64 encoded data. </base64>

The two compound types used by XML-RPC are *array* and *struct*. Array is a sequence of elements with the same or with different types. The *struct* data types are defined as pairs in form of name-value and are similar to hashtables.

For the purpose of this paper, we cannot allow the *type* attribute to take any valid XML value. For example, if we have an element with *type="decimal"*, we have no equivalent for that particular type in XML-RPC [6, 7]. Taking all this into consideration we will add a restriction to XRDL. The *type* attribute from a valid XRDL can have as a value either a simple data type that is also defined in the XML-RPC specification or a complex type that is defined in the types section of that XRDL document. We will prove in the next paragraphs that the compound data types from the XML-RPC specification can be defined in XRDL as a complex type using the simple types available.

2. XML-RPC described by a XRDL document

The first part of the equivalence is to prove the following Theorem:

Theorem 2.1. *Any XML-RPC can be characterized by a XRDL document.*

The characterization of a web service consists of describing the methods offered by that service. In order for a potential user to know how to call the methods, he needs to know the name of the function, the input of the function (the parameters) and the result that it returns [4]. XRDL has a simple way of describing the functions of a web service, defined in the following part of the XSD document:

```
<xs:element name="methods">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="method" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
                     <xs:complexType mixed="true">
                        <xs:attribute name="type" type="xs:string" use="required" />
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
               <xs:attribute name="result" type="xs:string" />
               <xs:attribute name="name" type="xs:string" />
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

From that schema we can see that any method can be described in XRDL and the demonstration in this case comes down to proving that any data type from XML-RPC has an equivalent in XRDL. Moreover, considering the restriction we put on the XRDL *type* attribute, we still have to prove that the compound types from XML-RPC can be defined in XRDL.

In the following Sections we will discuss the two compound data types *array* and *struct*.

The *array* type in XML-RPC contains an element named *data* that has one or more children. These children are each defined by a *value* tag which contains another tag specifiying the data type and the value. An array can be unidimensional if it contains only elements of simple data types or multi-dimensional if it also contains elements of other compound types.

Lemma 2.2. *A onedimensional array with n elements can be defined as a data type in XRDL,* $\forall n \in \mathbb{N}$.

We will prove Lemma 2.2 using an induction argument. For the case n=1, an array with one element will have the following general form:

```
<array>
   <data>
      <_simpleType>_customValue</_simpleType>
   </data>
</array>
```

_simpleType will be replaced by any of the types: string, int/i4, double,boolean, dateTime.iso8601 or base64 and *_customValue* will be replaced by a suitable value according to the data type chosen. In XRDL we define a complex data type:

```
<type name="_1_ElementArray">
   <member type="_simpleType"> _firstMember</member>
</type>
```

_1_ElementArray is the name given to the custom data type, *_simpleType* is the same type used in the XML array and *_firstMember* is the name given to the member of the array (Obs. The names given to the data type or to the members are irrelevant). As we can see this custom type is equivalent to the array given in XML. For the inductive step we assume that a onedimensional array with k elements can be defined as a complex type in XRDL. We have to prove that a onedimensional array with k+1 elements can be defined as a complex type in XRDL. Let the k+1-length array be:

```
<array>
   <data>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
      <value>
         <_simpleType_2>_value_2</_simpleType_2>
      </value>
      ...
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
      <value>
         <_simpleType_k+1>_value_k+1</_simpleType_k+1>
      </value>
   </data>
</array>
```

If we build an array containing only the first k elements, acoording to the induction hypothesis that array can be expressed as a complex type in XRDL as follows:

```
<type name="_k_ElementArray">
   <member type="_simpleType_1">_ member_1</member>
   <member type="_simpleType_2">_ member_2</member>
   ...
   <member type="_simpleType_k-1">_ member_k-1</member>
   <member type="_simpleType_k"> _member_k</member>
</type>
```

Now we can build a similar array with k+1 elements just by extending this type with one more element:

```
<type name="_k+1_ElementArray">
   <member type="_simpleType_1">_ member_1</member>
   <member type="_simpleType_2"> _member_2</member>
   ...
   <member type="_simpleType_k-1">_ member_k-1</member>
   <member type="_simpleType_k"> _member_k</member>
   <member type="_simpleType_k+1">_ member_k+1</member>
</type>
```

This is the k+1-length array that we had to transcribe in XRDL and in conclusion the hypothesis is proven for any $n \in \mathbb{N}$.

Now we will consider multi-dimensional arrays, which are arrays that contain other *arrays* or *struct* elements.

Lemma 2.3. *A n-dimensional array can be expressed in XRDL as a complex data type, for any* $n \in \mathbb{N}$.

An example of a two-dimensional array is:

```
<array>
   <data>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
      <value>
         <array>
            <data>
               <_simpleType_2>_value_2</_simpleType_2>
               <_simpleType_3>_value_3</_simpleType_3>
            </data>
         </array>
```

```
      </value>
   </data>
</array>
```

Before proving this hypothesis let us treat the second compound data type *struct*.

The *struct* type in XML-RPC contains a sequence of elements of type *member*. Each member has two children *name* and *value*. The *value* tag has another child that specifies the data type of that member and the actual value. A struct can also be multidimensional if it contains other elements of compund data types.

Lemma 2.4. *A onedimensional struct with n pairs of elements name-value can be defined in XRDL as a complex type, for any* $n \in \mathbb{N}$.

For the base case we define the general form of a struct with one pair of elements name-value:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType>_value_1</_simpleType>
      </value>
   </member>
</struct>
```

where *_simpleType* will be replaced by any of the types: string, int/i4, double, boolean, dataTime.iso8601 or base64. In XRDL we define a complex type accordingly:

```
 <type name=_1_PairStruct>
   <member type="string">_member_1</member>
   <member type="_simpleType">_value_1</member>
</type>
```

For the inductive step we assume that a onedimensional struct with k pairs of elements can be defined as a complex type in XRDL. We still have to prove that a onedimensional struct with k+1 pairs of elements can also be defined in XRDL. Let the struct with k+1 pairs of elements be:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </member>
   ...
```

```
   <member>
      <name>_member_k</name>
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
   </member>
   <member>
      <name>_member_k+1</name>
      <value>
         <_simpleType_k +1>_value_k+1</_simpleType_k+1>
      </value>
   </member>
</struct>
```

If we consider the first k pairs of this struct they would build another struct but with k pairs of elements, which according to the earlier assumption can be defined in XRDL as a complex type. Let that type be *_k_PairStruct*:

```
<type name="_k_PairStruct">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
   ...
   <member type="string">_member_k-1</member>
   <member type="_simpleType_k-1">_value_k-1</member>
   <member type="string">_member_k</member>
   <member type="_simpleType_k">_value_k</member>
</type>
```

Taking this into consideration we can build another complex type just by extending this type with two other members:

```
<type name="_k_PairStruct">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
   ...
   <member type="string">_member_k</member>
   <member type="_simpleType_k">_value_k</member>
   <member type="string">_member_k+1</member>
   <member type="_simpleType_k+1">_value_k+1</member>
</type>
```

And this is exactly the definition of our struct with k+1 pairs. So the hypothesis is proven for any $n \in \mathbb{N}$.

Now we go back to the demonstration of Lemma 2.3. For the base case we observe that a two-dimensional array would have to contain at least an element of a one-dimensional compound type and all the another elements can either be of a simple type or of a one-dimensional compound type. But we have already proven in Lemma 2.3 and Lemma 2.4 that any one-dimensional array

or struct can be defined in XRDL as a complex type. The next step would be to define another complex type in XRDL to build this two-dimensional array. The general form of a two-dimensional array in XRDL is:

```
<type name="_twoDimensionalArray">
   <member type="_type_1">_member_1</member>
   <member type="_type_2">_member_2</member>
   ...
   <member type="_type_m">_member_m</member>
</type>
```

where *_type_i* is a simple type or a one-dimensional compund type for each $i \in \{1, 2, ..., m\}$ and there is at least one $j \in \{1, 2, ..., m\}$ so that *_type_j* is a one-dimensional array or struct. The base case for the hypothesis in Lemma 2.3 is obviously true. For the inductive step we assume that any k-dimensional array can be defined in XRDL. We observe that a k+1-dimensional array has l-dimensional elements with $1 \leq l \leq k$. The inductive assumption says that for each element of this array we can define a complex type in XRDL, so for defining the k+1-dimensional array we can just build a complex type with elements of those types defined before.

For multi-dimensional struct we consider the following Lemma:

Lemma 2.5. *A n-dimensional struct can be expressed in XRDL as a complex data type, for any* $n \in \mathbb{N}$.

The exact same reasoning from Lemma 2.3 can be applied for proving this hypothesis with the observation that one member of the new added pair will always be of type string (the name of that pair) and the other member representing the value can be of any multi-dimensional compound type or of a simple type.

3. XRDL document as a valid XML-RPC service

This Section will consist of the demonstration for the following Theorem:

Theorem 3.1. *Any XRDL document describes a valid XML-RPC web service.*

As we have seen in Section 2 the structure of a XRDL document allows it to describe the methods of a web service and if necessary define compound data types. In XRDL the service methods are described by their name, input and output data. We can conclude that any method described is a valid XML-RPC method if the input and output data types are valid XML-RPC data types. Considering the restriction we added to XRDL in Section 1, any

simple type defined in XRDL will be a valid data type in XML-RPC. So we have to prove the following Lemma:

Lemma 3.2. *Any compound data type defined in XRDL is a valid XML-RPC type.*

For the proof of this Lemma we will assume that the XRDL document respects certain naming conventions specified in the following Proposition:

Proposition 3.3. *The XRDL compound types defined to describe the XML-RPC struct type will reflect the pairs of the struct by a naming convention: one member will always be of type string and its name will start with "_member" and the following member will be of any valid type and its name will start with"_value". The XRDL compound types defined to describe the XML-RPC array type will only contain members named with the word "_member".*

This Proposition ensures the fact that the compound data types defined in XRDL reflect the equivalent complex type from XML-RPC (*struct* or *array*). Otherwise this compound types could not be differentiated, since both XML-RPC complex types, *struct* and *array*, are transcribed in XRDL as a compound data type with an enumeration of members without any structural difference between them.

Considering Proposition 3.3 we can divide the demonstration of Lemma 3.2 into two parts, treating the compound types that contain members named with *"_member"* and *"_value"* separately from the other compound types.

The first case we treat is for compound types that contain members named with *"_member"* and *"_value"*. We consider the following Lemma:

Lemma 3.4. *A compound type from an XRDL document that contains members named with "_value" and has only members of a simple type can be transcribed in XML-RPC into an element of type struct.*

From the hypothesis of Lemma 3.4 we deduce that the compound element will have the general form:

```
<type name="_compoundElement_n">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
   ...
   <member type="string">_member_n</member>
   <member type="_simpleType_n">_value_n</member>
</type>
```

where *_simpleType_i*, for $\forall i = \overline{1,n}$, will be replaced by any of the types: string, int/i4, double, boolean, dataTime.iso8601 or base64.

For the proof of Lemma 3.4 we will use an induction for $n \in \mathbb{N}$. For the base case n=1 we have a compound element in the form of:

```
<type name="_compoundElement_1">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
</type>
```

This can be transcribed in XML-RPC as a *struct* element as follows:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </member>
</struct>
```

For the inductive step we assume that Lemma 3.4 is true for $n = k$, meaning a compound type with $2 * k$ members can be transcribed as a *struct* element. Let the compound type be:

```
 <type name="_compoundElement_k">
   <member type="string">_member_1</member>
   <member type="_simpleType_1">_value_1</member>
   ...
   <member type="string">_member_k</member>
   <member type="_simpleType_k">_value_k</member>
</type>
```

and the equivalent *struct* element:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </member>
   ...
   <member>
      <name>_member_k</name>
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
   </member>
</struct>
```

We observe that for the step k+1 the compound type will be:

```
<type name="_compoundElement_k+1">
   <member type="string">_member_1</member>
```

```
   <member type="_simpleType_1">_value_1</member>
   ...


   <member type="string">_member_k</member>
   <member type="_simpleType_k">_value_k</member>
   <member type="string">_member_k+1</member>
   <member type="_simpleType_k+1">_value_k+1</member>
</type>
```

Considering the equivalent *struct* for the compound type with $2 * k$ members we observe that in this case we can extend that struct by one more member in order to get an equivalent for this compund type. So we transcribe it into the following struct:

```
<struct>
   <member>
      <name>_member_1</name>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </member>
   ...
   <member>
      <name>_member_k</name>
      <value>
         <_simpleType_k>_value_k</_simpleType_k>
      </value>
   </member>
   <member>
      <name>_member_k+1</name>
      <value>
         <_simpleType_k+1>_value_k+1</_simpleType_k+1>
      </value>
   </member>
</struct>
```

This structure is a valid element in XML-RPC. This concludes the proof of Lemma 3.4.

As a generalization for Lemma 3.4 we have:

Lemma 3.5. *A compound type from an XRDL document that contains members named with "value" can be transcribed in XML-RPC into an element of type* struct.

We observe that Lemma 3.5 looses the restriction from Lemma 3.4 that specified that members can only be of a simple type. To prove Lemma 3.5 we

need to treat the cases of compound types that contain elements of another compound type.

Before proving Lemma 3.5 we need to treat the case of compound types that have equivalents in XML-RPC *array* elements. Let us consider the following Lemma:

Lemma 3.6. *A compound type from an XRDL document that contains only members named with the word "_member" and that has only members of a simple type can be transcribed in XML-RPC into an element of type array.*

Given the hypothesis of this Lemma the general structure of the compound element is:

```
<type name="_n_compoundArrayElement">
   <member type="_simpleType_1">_member_1</member>
   ...
   <member type="_simpleType_n">_member_n</member>
</type>
```

where *_simpleType_i*, for any $i = \overline{1,n}$, will be replaced by any of the types: string, int/i4, double, boolean, dataTime.iso8601 or base64. Let us prove this Lemma by applying an induction on n, the number of members of the compound type. For the base case $n = 1$ we have the following compound type:

```
<type name="_1_compoundArrayElement">
   <member type="_simpleType_1">_member_1</member>
</type>
```

This can be described in XML-RPC as an *array* element as follows:

```
<array>
   <data>
      <value>
         <_simpleType_1>_value_1</_simpleType_1>
      </value>
   </data>
</array>
```

For the induction step we assume that a compound element with k members can be described in XML-RPC as an *array* element and prove that we can also describe a compound element with k+1 members. Let the compound element with k members be:

```
<type name="_k__compoundArrayElement">
   <member type="_simpleType_1">_member_1</member>
   ...
   <member type="_simpleType_k">_member_k</member>
</type>
```

and let its equivalent in XML-RPC be:

```
<array>
  <data>
    <value>
      <_simpleType_1>_value_1</_simpleType_1>
    </value>
    ...
    <value>
      <_simpleType_k>_value_k</_simpleType_k>
    </value>
  </data>
</array>
```

We have to prove that we can find an equivalent array element in XML-RPC for a compound type from XRDL with k+1 members. Let that compound type be:

```
<type name="_k__compoundArrayElement">
  <member type="_simpleType_1">_member_1</member>
  ...
  <member type="_simpleType_k">_member_k</member>
  <member type="_simpleType_k+1">_member_k+1</member>
</type>
```

We observe that, considering only the first k members of this compound type, we can build an array in XML-RPC given the previous assumption. We can easily extend that array by one element to obtain an equivalent array for the compound type with k+1 members:

```
<array>
  <data>
    <value>
      <_simpleType_1>_value_1</_simpleType_1>
    </value>
    ...
    <value>
      <_simpleType_k>_value_k</_simpleType_k>
    </value>
    <value>
      <_simpleType_k+1>_value_k+1</_simpleType_k+1>
    </value>

  </data>
</array>
```

This concludes the induction that proves Lemma 3.6.

Now we can go back to prove Lemma 3.5. As we mentioned before, in order to prove Lemma 3.5, we need to demonstrate that compound types containing

members named with the word *"_value"* and that contain elements of another compound type have equivalents in XML-RPC as an element of type *struct*.

We observe that a compound type element can include recursively members of other compound types. In the following example *_1_compoundArrayElement1* includes *_1_compoundArrayElement2*:

```
<type name="_1_compoundArrayElement1">
  <member type="_1_compoundArrayElement2">_member_1</member>
</type>
<type name="_1_compoundArrayElement2">
  <member type="_simpleType_1">_member_1</member>
</type>
```

We define the dimension of a compound type as the number of compound types involved recursively (including the defined compound type itself) in the definition of that particular compound type. In the previous example *_1_compoundArrayElement1* has dimension 2 and *_1_compoundArrayElement2* has dimension 1.

The proof for Lemma 3.5 can be understood as an induction on $n \in \mathbb{N}$, where n represents the dimension of the compound type. We observe that Lemma 3.4 can be seen as a particular case for Lemma 3.5 with $n = 1$. So the base case for the induction is proven by Lemma 3.4. Furthermore, we know that in XML-RPC we can recursively define elements of type *struct* with any finite dimension. Considering the previous observations a simple induction step proves that any k-dimensional compound type that respects the hypothesis in Lemma 3.5 can be described in XML-RPC as a k-dimensional element of type *struct*. This concludes the proof for Lemma 3.5.

Finally, we consider the following Lemma that treats the general case for Lemma 3.5 without the restriction that members can only be of a simple type.

Lemma 3.7. *A compound type from an XRDL document that contains only members named with the word "_member" can be transcribed in XML-RPC into an element of type array.*

The exact same reasoning used in proving Lemma 3.5 can be applied for proving Lemma 3.7 with the observation that the XML-RPC type *array* can be n-dimensional for any $n \in \mathbb{N}$.

4. Conclusions and future work

Sections 2 and 3 demonstrate that any XML-RPC document can be described by a XRDL document and any XRDL document that respects the restrictions mentioned in this article will also describe a valid XML-RPC service. This leads to the conclusion that XRDL is a valid Description Language for XML-RPC. The higher motivation of this demonstration is to be able to

use XRDL for Web Service Matching [10] and for automatic generation for servers and clients written in different programming laguages.

In a future article we plan to propose an extension to the XSD of XRDL, so that it reflects the restrictions imposed on the *type* attribute in Section 1 of this article.

Many researchers still argue the utility of XRDL as a Description Language for XML-RPC web services. This paper started to prove its utility by proving it is a valid Description Language for XML-RPC services.

For future work we plan to exploit further utilities of XRDL. We will study ways of generating the XRDL description automatically[1]. Furthermore we will find ways of generating a XML-RPC service skeleton using the corresponding XRDL description and ways of generating a client that calls the methods of the XML-RPC service described in a XRDL document[2].

References

[1] F. Boian, B. Jancso, Uniform Solutions for Web Services, Studia Univ. Babe-Bolyai, vol. 57, no. 3, 2012
[2] F. Boian, D. Chice, D. Ciupeiu, D. Homorodean, B. Jancso, A. Ploscar, WSWrapper A Universal Web Service Generator, Studia Univ. Babe-Bolyai, vol. 55, no. 4, 2010
[3] XRDL Project Home, http://code.google.com/p/xrdl/
[4] XML-RPC Specification, http://xmlrpc.scripting.com/spec.html
[5] XML-RPC Data Model, http://www.tutorialspoint.com/xml-rpc/xml_rpc_data_model.htm
[6] XML-RPC, Ken Slonneger, 2006, http://homepage.cs.uiowa.edu/ slonnegr/xml/10.XML-RPC.pdf
[7] XML Schema Tutorial, http://www.w3schools.com/schema/
[8] XML-RPC, http://en.wikipedia.org/wiki/XML-RPC
[9] XML Matters: XML-RPC as object model, David Mertz, 2001, http://www.ibm.com/developerworks/xml/library/x-matters15/index.html
[10] F.M. Boian, A. Ploscar, R.F. Boian, Web Service Matching, Knowledge Engineering Principles and Techniques, Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2013 Cluj-Napoca (Roamania), July 4-6, 2013, this volume

(1) Babeş-Bolyai University, Department of Computer Science, Cluj-Napoca, Romania

E-mail address: dianatroanca@yahoo.com

E-mail address: florin@cs.ubbcluj.ro