Schema Publishing in X.500 Directory

Rajeev P.V. <rajeev@iitm.ernet.in> *
S.V. Raghavan <svr@iitm.ernet.in> *
Glenn Mansfield <glenn@aic.co.jp> †

* Department of Computer Science & Engineering
Indian Institute of Technology, Madras 600036
INDIA.
† AIC Systems Laboratory, Sendai, Japan

Abstract

The X.500 directory provides a powerful mechanism for storing and retrieving information about objects of interest. To interpret the information stored in the directory, the schema must be known to all the components of the directory. The rapid increase in the number of schemas and the relatively slow pace of standardization and subsequent publication has given rise to the challenge of making the schema accessible to all the components of the directory. In this paper, we address this issue and present a strategy using the existing mechanisms of the directory. We present a naming scheme for naming schema objects, a meta-schema for storing schema objects. We discuss the procedures for bootstrapping the DSAs by loading the schema from the directory and for fetching unknown schema from the directory at runtime. The procedures involve an algorithmic retrieval of schemas from the directory. The algorithm used is presented and the design details are discussed. The actual implementation is under progress.

I. Introduction

The X.500 directory standard provides a powerful mechanism for storing and retrieving information about objects of interest in the networking world [2]. This information is distributed and managed across a network of many autonomous sites. Directory is now used for a wide range of applications starting from name/address lookup to network management, from restaurant information to bibliographic information services. New applications keep coming up. Though application designers (are expected to) use the standardized schema rules for objects that need to have entries in the directory, the novelty of the applications has resulted in the generation of a host of new schema rules. In order to interpret the information stored in the directory, the components of the directory must have knowledge about the structure and representation (schema) of the information held within the directory. Further, with the increase in the application of directory to the real world problems, changes to the schema will need to be taken into active consideration.

The distributed nature of the network, and the relatively slow process of standardization, have given rise to the challenging task of making accessible the information about the schema rules itself. A mechanism for propagating the schema among the functional components of the directory is required.

In this paper we address the issue of schema publishing and propose a strategy for the implementation using the existing mechanisms of the directory itself. We present a naming scheme for naming schema objects and a meta-schema for storing schema objects. We discuss the procedures for bootstrapping the Directory System Agents (DSAs) by loading the schema from the directory and fetching unknown schema from the directory itself at runtime. The rest of the paper is organised as follows. Section II discusses the present status of directory. In section III the storage of schema information in the directory is discussed. Section IV presents an algorithm for retrieval of schema from the directory. In section V, we present a schema publication strategy. The behaviour of directory components are discussed in section VI and section VII. The details of implementation are presented in section VIII. Finally, section IX spells out the conclusion.

II. The Directory - Present Status

The X.500 directory deployment is proceeding at a fast pace. Presently there are more than 450 DSAs spanning 35 nations covering a total of more than a million entries. The growth rate

is found to be significant with about 150 DSAs added every year since November 1990 [3]. Directory is now being used for a variety of applications in distributed computing environment. It is used in distributed systems [9], network management [7,8,6,10], mailing list generation and for storing bibliographic information [1].

The storage and retrieval mechanism provided by the directory is powerful and flexible. However, the key to the directory is the knowledge of the schema rules defined for the objects represented in the directory. To facilitate the diffusion of this knowledge appropriate schema management mechanisms need to be designed. Schema management involves :

- Storage of schema information in the directory.
- Definition of rules for schema modification.
- Algorithmic access/retrieval of schema information in the directory
- Propagation of schema information from one component of the directory to other components of directory.

In [4], a mechanism for the storage of schema entries is described. It also describes a naming mechanism for resolving unknown schema. In the proposed method a single object stores the information about the entire subschema stored in a DSA. The proposal does not mesh well with the distributed concept of the directory. As more and more schema objects are represented in the directory, the subschema object proposed in [4] will become very large and at some stage it may become too big to handle.

The paper [5] on schema management proposes a few solutions for schema management and propagation. For enabling the Directory User Agent (DUA) to resolve unknown schema, one of the solutions proposed is to transmit schema with every query response. This is obviously an inefficient approach because of the additional overhead involved in the query processing. Another solution suggested is to increase the functionality of the DUA and use existing Directory Access Protocol (DAP) to retrieve the schema information. The proposal does not specify any standard naming scheme for locating the subschema entries. Another solution proposes an enhancement of the DAP functionality. This requires a change in the DAP. For propagating the schema between DSAs, a new protocol is suggested. A second method suggests the use of the directory replication protocols to replicate the subschema subentry.

The strategy proposed in this paper has a very simple and workable approach. No added DAP/DSP functionality is envisaged. At the same time the scalability problems are avoided. In essence, it allows the distributed storage of schema objects and provides a schema propagation method based on the distributed storage mechanism. As schema objects are stored as separate entries in the directory, the update and modification operations are much simpler. Of course, on the down side, more than one directory read operation *may* be required to retrieve the information about objects and its attributes as objects and attributes are stored as separate entries in the directory.

As schema information of all objects in a naming context are stored below the starting entry of that naming context, the same DSA will be able to supply the schema information stored in that DSA. Thus there is no need to contact another DSA for resolving the schema of an object stored in the local DSA.

This method does not require complex attribute syntaxes to store schema information. Attribute syntaxes defined in [4] are quite complex and thus new attribute syntaxes must be defined to store and interpret those attributes. The attributes used in our proposal are defined in terms of attribute syntaxes that are widely used [2,11] and thus can be easily implemented.

III. Storage of Schema Information in the Directory

The primary step towards the schema management is to store the schema information in some format in the directory. Storing schema information in the directory has certain obvious advantages:

- The components of the directory can access the schema information using the standard directory protocols.
- The distributed nature of the directory allows the schema also to be distributed. Schema used locally can be kept in the local DSA itself whereas schema of general objects like person, organization etc can be made available to all components of the directory by publishing it.

III.A Naming Scheme for the Schema Objects

The schema information is stored in a distributed manner. We propose a model in which each naming context stores the schema relevant to it. Schema of objects which are commonly used will be stored in first level DSAs and will be replicated to DSAs below it. (This results in all DSAs having a copy of the schema of objects that are commonly

used.) Schema defined in any DSA will be replicated to all its subordinates. Thus a DSA at the lowest level will have a replicated copy of schema stored in all its superior DSAs. But the schema is not propagated to a higher level DSA. Thus local schema will not be propagated to DSAs above the naming context where it is defined, but it will be propagated to subordinate DSAs.

To store the schema information, an object called *subschema* object is defined. This object can come anywhere in the Directory Information Tree (DIT). The subschema is defined as a subclass of *Top*. The subschema entry is stored below the starting entry of a naming context. Starting entry of a naming context must contain a subschema subentry, and its name should be *CN= Subschema* as specified in [4] . This standard naming methodology is necessary so that the components of the directory can easily locate the schema entries. All schema information relevant to that naming context is stored below the subschema entry. Children of the subschema entry store information about objects, attribute types, attribute syntaxes or matching rules. The DIT structure for storing schema information is shown in Figure 1.

III.B Meta-Schema for Storing Schema Information

The information about objects, attribute types, attribute syntaxes and matching rules are stored as separate entries. Schema for these objects are given in the appendix.

This schema can be easily implemented. We used this method to store the schema in the directory. The meta-schema is known to the components of the directory and thus they can load the schema from the directory.

IV. Retrieval of Schema from the Directory

When an unknown object is encountered by any component of directory during a directory operation, it proceeds the following way to resolve the schema.

The last RDN component in the name of the object whose schema is to be resolved is removed and the RDNs *CN=subschema\ oid= object identifier of the new object* is added to the name. A read request is initiated for the name so formed. If the entry is not found, the last and second last RDNs of the name of the entry are removed and the RDNs *CN=Subschema\ oid= object identifier of the new object* is added to the name. For example, let the name of the object whose schema is to be resolved be *\C=India \O=Indian Institute of Technology, Madras \OU=Department of Computer Science\IPNI=spark*. If the schema of the object *IPNI* (IP Node Image) is not known to a component of the directory, it will adopt the following procedure.

Let the object id for the object *IPNI* be *ipni*. The last RDN (*IPNI=spark*) is removed from the distinguished name of the entry and the RDNs *CN= Subschema \oid=ipni* is appended. The name thus formed is *\C=India \O=Indian Institute of Technology, Madras\OU= Department of Computer Science \CN= subschema \oid=ipni*. A read request is initiated on this name. If the distinguished name *\C=India \O=Indian Institute of Technology, Madras \OU= Department of Computer Science* is the context prefix of a naming context, this read request will result in the directory returning the schema for the object IPNI. If it is not, the read operation will fail. If the read operation returns error, the second last RDN (*OU=Department of Computer Science*) is removed and again a read operation is initiated with distinguished name *\C=India \O=Indian Institute of Technology, Madras \CN=subschema \oid=ipni*. For the DIT structure shown in Figure 1, this query will succeed and returns the schema information. The schema for the requested object will always be located below the starting entry of the naming context in which the entry is located. The method explained here is presented as an algorithm in Figure 2.

V. Publication of Schema

Once the issues of storing schema information and resolving the schema information are addressed, we have to see how these schema information can be actually distributed. The distribution of the schema should be in such a way that the algorithm explained above can be directly used.

For the publication of the schema following procedure has to be adopted. When an object is defined in any naming context, propagate the schema information for that object to all DSAs holding the subordinate naming contexts. If the DSA is at the first level, propagate it to all first level DSAs. Each DSA receiving the schema information from its superior DSA (or from another first level DSA if it is a first level DSA) it will propagate the received schema to the DSAs storing the subordinate naming contexts. Thus all DSAs storing the subordinate naming contexts of the naming context where the new object is defined, will get a copy of the schema object. Similarly, the modification operations should be propagated to the subordinate naming contexts. The DSA has to identify the modify

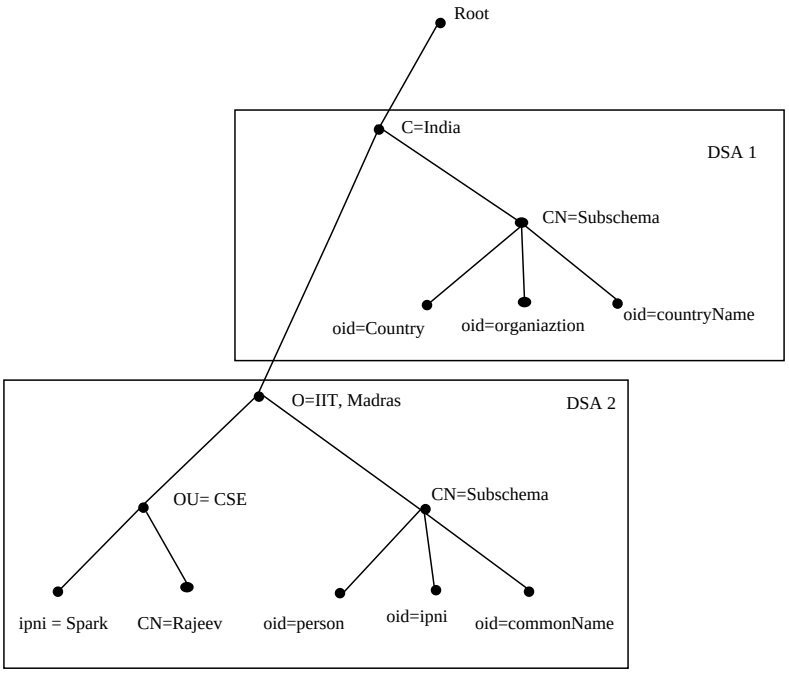

Figure 1: DIT structure for storing schema information.

operations involving schema objects and propagate them to its subordinates.

Locally defined schema in the DSAs lower in the hierarchy will not be propagated to its superior DSAs. This avoids the massive replication of local schema. This is necessary as the local schema may be modified quite often and massive replication of local schema make schema modification difficult.

VI. DSA Behavior

The behavior of the DSA can be summarised as follows. To perform the schema management the entire schema of the objects recognized by that DSA must be stored in its memory. Each DSA maintains a schema table which contains information about objects, attribute types, syntaxes matching rules etc. This table has to be loaded from the directory at startup and modifed when a schema modification request is received. The procedures for startup and fetching schema during runtime are given below. Once the schema information is in the memory, DSA can perform all schema checks for maintaining the integrity of the data held in the directory.

VI.A Startup

On startup, DSA looks for the schema entries in the directory. DSA scans the knowledge base and locate all locally held naming contexts. Then it appends the name $CN{=}Subschema$ to all context prefixes and issue a local list command with these names as the base entry. Then a read request is issued on all entries obtained on list request. This way the complete entries can be loaded into the schema table of the DSA. A naming context indicator is also included in the schema table so that the DSA can ensure that it is checking the right schema when multiple naming contexts are present in a single DSA.

VI.B Runtime

When a DSA encounters an object or an attribute type whose schema is unknown to it, it generates a read request to obtain the information about the schema of that object. The DSA follows the procedure explained in the Figure 2. When the DSA receives the schema information, it checks whether the information within that schema is also known to the DSA. For eg. while retrieving the schema of an object, DSA has to check that all attribute types used by that object is also known to the DSA and also whether the object classes used in the object class hierarchy of the object is known to the DSA. If any of these information is unknown to the DSA, it again sends a read request to get the schema information about those objects.

```
Algorithm FindSchema (TargetEntry)
/* This algorithm returns the schema of the object class
of the entry given by TargetEntry.*/
1.  temp = TargetEntry
    temp1 = TargetEntry
    TargetObjectID = ObjectidOf (LastRDNOf (TargetEntry))
2.  temp1 = temp1 - LastRDNOf (temp1)
     /* remove the last rdn of the entry */
    if NumberOfRDNs (temp1) == 0 goto 4
     /* entry not present in the directory - a fatal error which
      * should not occur. The only chance of this
      * is when the presented name (TargetEntry)
      * wrong as the schema information for a given
      * should be present in the directory before
      * entry can be added to the directory */
    SchemaObject = temp1 + CN=Subschema + oid=TargetObjectID
     /* append the name of the schema entry */
3.  Read ( SchemaObject, schema )
     /* issue a read operation to the directory */
    if failure goto 2
     /* not found, so go higher up in the hierarchy */
    return schema
     /* the schema for the object is located */
4.  return error
     /* The supplied name is wrong as the schema information for
     the object should be present in the directory somewhere
      above the entry of the object itself.
     */
```

Figure 2: An Algorithm for Locating Schema Subentry in the Directory.

VII. DUA Behavior

The DUAs starts by loading the schema information stored in its local oid tables. The local oid table contains the most commonly used schema information. When the DUA encounters an entry with unknown schema, it follows the algorithm explained in Figure 2. The last RDN of the distinguished name is removed and the name *CN=Subschema* is appended and a read request is sent to the directory. If this fails, one more RDN is removed and another request is sent to the directory. This is repeated until the request succeeds. The schema thus obtained is then loaded into the memory structures for interpreting the information stored in the directory.

VIII. Implementation Details

The implementation of the scheme discussed in this paper is done in the public domain implementation of X.500, Quipu which is a part of ISODE-8.0 [11,12]. Quipu stores the schema information in a set of flat files. The Quipu code was modified to enable it load schema from the directory itself. The DSA used in our implementation is managing the Indian Institute of Technology, Madras portion of the DIT. The schema information required by the DSA *@c=IN@cn=Panda* (the DSA administering the IIT Madras portion of DIT) is stored in the directory below the entry *@c=IN@o=Indian Institute of Technology Madras@cn=Subschema.*

VIII.A Loading Schema from the Directory During Startup

Quipu DSA is modified to load the schema from the directory. Quipu DSA reads the tree-directory parameter from the quiputailor file and opens the EDB file stored in the tree directory. It scans the EDB files and checks for entries with RDN *CN=Subschema*. If an entry with that name is found, it goes down the directory storing the subordinates of the subschema entry and loads the schema into the oid tables. If no subschema entry is found in the first level, it goes down to the subdirectories which stores the subordinates of the first level entries and continues searching. This procedure is done recursively until the entire database is scanned and the schema loaded to the directory.

Once the EDB file storing the schema entries are located, the DSA starts loading the schema to the oid tables. The EDB file is first scanned for entries storing attribute types. Loading of object class schema involves multiple scanning of the EDB file. While loading the object class hierarchies, objects in the root of the object class hierarchy must be loaded first and then the subordinates and so on. Even if we keep the entries in the directory arranged in the order of object class hierarchy, DSA sorts the entries and thus the order of entries will

be changed. Thus multiple scanning of the EDB file is required.

The timings are measured for the DSA @c=IN@cn=Panda to load schema from the directory. This DSA stores 231 attribute types and 90 object classes in its subschema subentry. The time required to load these subschema entries to the memory during startup is found to be 41.97 seconds. The startup time mainly depends on the number of attribute types and object classes stored in the subschema subentry.

VIII.B Fetching Schema from the Directory During Runtime

The DUA *dish* of quipu loads the schema from the oid tables. When it encounters an object with unknown schema, it sends a request to the DSA by following the algorithm given in the Figure 2. The schema entry retrieved from the directory is loaded into the internal structures of the dish. Once the schema is loaded dish proceeds with the operation requested by the user. The DSA behavior is also identical.

The time required to perform the schema resolution are measured. The schema resolotion time depends mainly on the location of subschema subentry in the directory. In a particular case, where the object was located at *@c=IN @ o=Indian Institute of Technology Madras @ ou=Computer Science and Engineering @ ipni=Spark* and the schema information was stored below *@c=IN @ o=Indian Institute of Technology Madras @ cn=subschema.* This case require two read operations because the schema resolution algorithm first forms the name *@c=IN @ o=Indian Institute of Technology Madras @ ou=Computer Science and Engineering @ cn=Subschema @ oid = ipni.* This will fail and then a new name is formed after removing the RDN *ou=Computer Science and Engineering.* This will succeed and returns the schema object. This experiment was carried out in a LAN and the time taken was found to be 0.6 seconds.

IX. Conclusion

We presented a data model for storing the meta schema in the directory and a naming scheme for naming schema objects. An algorithm for locating schema in the directory based on this naming scheme and data model is presented for resolving unknown schema. Loading the schema from the directory during the DSA bootstrapping and fetching the schema from the directory in runtime is also explained. The merit of the strategy lies in its simplicity. No added DAP/DSP functionality is envisaged. It builds on the existing directory mechanisms. At the same time by distributing the schema information the issues of scalability are suitably addressed. Implementation experience is discussed. Presently experimentation is done locally. Real wide-area distributed experimentation is planned for the coming months.

X. References

[1] Paul Barker. *DIT Structures to support searching of Bibliographic information and information resources.* University College London, December 1992.

[2] CCITT. *The Directory - Overview of Concepts, Models and Services, Recommentations X.500 - X.521* 1988.

[3] David Goodman,*Paradise International Report 4.* Paradise Pilot Group, University College London, November 1992.

[4] Tim Howes, *Schema Information in the X.500 Directory*, Internet Draft, Work in Progress, University College Michigan, July 1992.

[5] Daniel L. Silver, James W. Hong, Michael A. Bauer *X.500 Schema Management*, University of Western Ontario, June 1993.

[6] Thomas Johannsen, Glenn Mansfield, Mark Kosters, Srinivas Sataluri *Representing IP Information in X.500 Directory.* RFC 1608, September 1993.

[7] James W Hong, Micheal A Bauer, J Michael Bennet. *Integration of Directory Services in the network management framework.* Third international symposium on integrated network management, April 1992.

[8] James W Hong, Micheal A Bauer, J Micheal Bennet, *Integration of Directory Services in distributed systems management.* International conference on parallel and distributed systems, 1992.

[9] Robert Van Rennesse, Andrew S Tanenbaum. *A Directory service supporting availability and consistancy.*Vrije Universiteit, Amsterdam.

[10] Glenn Mansfield, Thomas Johannsen, Mark Knopper, *Charting networks in the Directory*, RFC 1609, February 1993.

[11] S.H. Kille. *The Design of Quipu.* Technical Report RN/89/19, University College London, March 1988.

[12] Colin J. Robins, Stephan E Kille, *The ISO Development Environment: User's Manual. Volume 5: Quipu.* X-Tel services, University College London, July 1991.

Author Information

Rajeev P.V. obtained his Bachelor's degree in 1991 from the University of Calicut, India in the field of Computer Science and Engineering. He is doing his Master's in Computer Science and Engineering in Indian Institute of Technology, Madras, India. He is also working as a research fellow in Networks Systems Laboratory, Department of Computer Science, Indian Institute of Technology, Madras, India. His research interests are networks and multimedia systems.

S.V. Raghavan is on the faculty of the Department of Computer Science and Engineering, Indian Institute of Technology, Madras. He is also the Chief Investigator of the project on *Education and Research in Computer Networking* jointly sponsored by the Department of Electronics, Government of India and the United Nations Development Programme. He is a life member of the Computer Society of India, a member of the Institution of Engineers and a fellow of institute of Electronics and Telecommunication Engineers. He is presently serving on the Board of Editors of the journal of IETE for computers and control. He is also a member of the Editorial Advisory Board for Computer Communications, Butterwoth-Heinemann Ltd. His research interests are networks, protocols, multimedia systems and performance.

Glenn Mansfield obtained his Masters degree in 1977 from Indian Institute of Technology, Kharagpur, India in the field of nuclear and particle physics followed by his Masters in physical engineering in 1979 from the Indian Institute of Science, Bangalore, India. After working with Tata Consultancy Services, Bombay, India as a Senior Systems Analyst, for five years he obtained his PhD specialising in logic programming from Tohoku University Japan. He has worked as a research associate in the computer center of Tohoku University, for a period of three years and is currently a senior visiting researcher at AIC systems Laboratories, Sendai, Japan. His areas of interest includes expert systems, logic programming, computer networks and their management. He is a member of Internet society, the IEEE, IEEE communications society and the Information Processing Society of Japan.

Appendix

```
subschema OBJECT CLASS
        Subclass of TOP
            MUST CONTAIN {
                commonName                  caseIgnoreString
                                - - For naming
            }

ObjectClass OBJECT CLASS
        Subclass of TOP
            MUST CONTAIN {
                objectIdentifier            [0] OBJECT IDENTIFIER,
                    - - This field stores the object identifier of object
                    - - represented by an object class entry. This attribute
                    - - is used for naming an object class entry.
            }
            MAY CONTAIN {

                    commonName                  [0] caseIgnoreString,
                                                - - This field is used to store the name of the object
                    mandatoryNamingAttributes   [1] SET OF OBJECT IDENTIFIER,
                    mandatoryAttributes         [0] SET OF OBJECT IDENTIFIER,
                    optionalNamingAttibutes     [1] SET OF OBJECT IDENTIFIER,
                    optionalAttributes          [2] SET OF OBJECT IDENTIFIER,
                    obsolete                    [3] BOOLEAN DEFAULT FALSE,
                    description                 [4] caseIgnoreString,
                    subClassOf                  [5] SET OF OBJECT IDENTIFIER

            }

AttributeType OBJECT CLASS
        Subclass of Top
            MUST CONTAIN {
                objectIdentifier            [0] OBJECT IDENTIFIER,
            }
            MAY CONTAIN {

                        commonName      [0] caseIgnoreString,
                                        - - used to store the name of the attribute type
                        constraint      [1] Constraint,
                        attributeSyntax [2] OBJECT IDENTIFIER,
                        multivalued     [3] BOOLEAN DEFAULT FALSE,
                        obsolete        [4] BOOLEAN DEFAULT FALSE,
                        matchRules      [5] SET OF OBJECT IDENTIFIER,
                        description     [6] caseIgnoreString

        }

AttributeSyntax OBJECT CLASS
        Subclass of Top
            MUST CONTAIN {
                objectIdentifier    [0] OBJECT IDENTIFIER
            }
            MAY CONTAIN {
                                    commonName      [0] caseIgnoreString,
                                                    - - Name of the attribute syntax
                                    datatype        [1] ASN1DataType,
                                    description     [2] caseIgnoreString,
                                    obsolete        [3] BOOLEAN DEFAULT TRUE

        }

   MatchingRule OBJECT CLASS
        Subclass of Top
            MUST CONTAIN{
                objectIdentifier    [0] OBJECT IDENTIFIER
            }
            MAY CONTAIN{
                                    commonName      [0] caseIgnoreString,
                                    matchtype       [1] MatchingTypes,
                                    description     [2] caseIgnoreString
                                    obsolete        [3] BOOLEAN DEFAULT TRUE

            }
   MatchingTypes ::= INTEGER {
                                        PRESENT                 (0),
                                        EQUALITY                (1),
                                        ORDERING                (2),
                                        CASESENSITIVEMATCH      (3),
                                        CASEINSENSITIVEMATCH    (4)

        }
```

```
Constraint ::=Choice {
        stringConstraint    [0] SizeConstraint,
        integerConstraint [1] Range
}
SizeConstraint ::= SEQUENCE {
        shortest INTEGER,
        longest INTEGER OPTIONAL
}
Range ::= SEQUENCE {
        lowerbound INTEGER,
        upperbound INTEGER
}
```