Handling Hierarchically Structured Resources Addressing Interoperability Issues in Digital Libraries

Maristella Agosti, Nicola Ferro, and Gianmaria Silvello

Abstract. We present and describe the NEsted SeTs for Object hieRarchies (NESTOR) Framework that allows us to model, manage, access and exchange hierarchically structured resources. We envision this framework in the context of Digital Libraries and using it as a mean to address the complex and multiform concept of interoperability when dealing with hierarchical structures. The NESTOR Framework is based on three main components: The Model, the Algebra and a Prototype. We detail all these components and present a concrete use case based on archives that are collections of historical documents or records providing information about a place, institution, or group of people, because the archives are fundamental and challenging entities in the digital libraries panorama. Within the archives we show how an archive can be represented through set data models and how these models can be instantiated. We compare two instantiations of the NESTOR Model and show how interoperability issues can be addressed by exploiting the NESTOR Framework.

1 Introduction and Motivation

An important challenge in the research work on Digital Libraries is to transform them in a new type of information infrastructures that can be user-centered, able to support content management tasks together with tasks devoted to communication and cooperation. That is information infrastructures that become common vehicle by which every user can access, discuss, evaluate, and enhance information of all forms. Although they are still places where information resources can be stored and made available to end users, the current design and development efforts are moving

Maristella Agosti · Nicola Ferro · Gianmaria Silvello
Department of Information Engineering, University of Padua, Italy
Tel.: +39 049 827 7650; Fax: +39 049 827 7799
e-mail: agosti@dei.unipd.it
Tel.: +39 049 827 7939; Fax: +39 049 827 7799
e-mail: ferro@dei.unipd.it
Tel.: +39 049 827 7929; Fax: +39 049 827 7799
e-mail: silvello@dei.unipd.it

in the direction of transforming them into infrastructures able to support the user in different information centric activities. In the context of digital libraries we need to take into account several distributed and heterogeneous information sources with different community backgrounds such as libraries, archives and museums and different information objects ranging from full content of digital information objects to the metadata describing them. These objects can be exchanged between distributed systems or they can be aggregated and accessed by users with distinct information needs and living in different countries. Digital Libraries are heterogeneous systems with peculiarities and functionalities that range from data representation to data exchange passing through data management. Furthermore, Digital Libraries are meaningful parts of a global information network which includes scientific repositories, curated databases and commercial providers. All these aspects need to be taken into account and balanced to support final users with effective and interoperable information systems.

A common goal in the design and development of Digital Libraries is to build systems which rely as much as possible on existing building blocks, thus maximizing the exploitation of Web and Internet standards. This trend is evident if we consider the standard technologies and protocols adopted over the years by Digital Libraries. Two of the main technologies of choice in Digital Libraries are: the eXtensible Markup Language (XML)[1] and the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)[2]. These technologies are very strongly interlinked with the Internet and the Web: XML is the technology of choice representing and encoding metadata in Digital Libraries. It was originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly fundamental role in the exchange of a wide variety of data on the Web. OAI-PMH is the standard *de-facto* for metadata exchange in distributed environments and its basic functioning is based on the Internet infrastructure (e.g. OAI-PMH requests are based on HyperText Transfer Protocol (HTTP) requests) [30].

These technologies are designed and shaped to be used within the Digital Libraries but at the same time their scope is fairly broad and they can be used within a large corpus of resources and systems. For these reasons they are either the means for addressing interoperability or the possible sources of interoperability issues. Indeed, on the one hand, we can build on these technologies and exploit them to constitute an integrated framework which handles the heterogeneity of Digital Library resources and functionalities. On the other hand, they can constitute a barrier towards the very interoperability they are aiming to foster. When it comes to modeling, managing, accessing and exchanging the resources of interest, often the design of the models and systems is driven by the technology-of-choice characteristics, thus they are bound to the technologies. A fundamental step is to define an organic and general framework free from specific technologies; in this way the technologies of choice can be

[1] `http://www.w3.org/XML/`
[2] `http://www.openarchives.org/pmh/`

conveyed into a well-defined path where their characteristics can be exploited to the maximum to address interoperability and to meet users requirements.

The difficulty in accomplishing this goal is due to the very nature of interoperability as a complex and multiform concept, which can be defined - as by the "ISO/IEC 2382-01, Information Technology Vocabulary, Fundamental Terms" - as follows: "*The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units*". When the concept of interoperability is considered in the context of digital libraries it takes on different dimensions as it has been evidenced by the *European Commission Working Group on Digital Library Interoperability* [13] which has identified six dimensions that can be distinguished and taken into account:

Interoperating entities. These can be assumed to be the traditional cultural heritage institutions, such as libraries, museums, archives, and other institutions in charge of preservation of artifacts or that offer digital services.

Information objects. The entities that actually need to be processed in interoperability scenarios. Choices range from the full content of digital information objects to the metadata describing them.

Functional perspective. This may simply be the exchange and/or propagation of digital content. Other functional goals are aggregating digital objects into a common content layer.

Multilingualism. Linguistic interoperability can be thought of in two different ways: as multilingual user interfaces to digital library systems or as dynamic multilingual techniques for exploring the digital library systems object space.

User perspective. Interoperability concepts of a digital library system manager differ substantially from those of a content consuming end user.

Interoperability technology. Enabling different kinds of interoperability constitutes a major dimension and several technologies designed in the context of Digital Libraries such as OAI-PMH and the Dublin Core[3].

The dimensions of interoperability are often analyzed and addressed focusing on a specific one; e.g. we can consider interoperability between the Digital Libraries entities with their information objects, we can consider the functional perspective of Digital Libraries by focusing on the exchange of objects or we can take into account only the cross-language access of information objects. Our aim is to consider all six dimensions of interoperability and define a common framework that can address all of them; to do so, we consider one of the most diffuse and important resources – i.e. hierarchically structured resources; in particular we employ a meaningful and challenging reality: archives. Archives are one of the main organizations of interest for Digital Libraries; they are a meaningful example of the need to support document management and access, as well as interoperability among the systems that manage different co-operating and related archives. The fundamental characteristic of archives resides in their internal hierarchical organization that constitutes both a

[3] `http://www.dublincore.org/`

challenge for their representation, managing and, exchange and a relevant feature for addressing interoperability.

In this chapter we present a conceptual and logical framework called “NEsted SeTs for Object hieRarchies (NESTOR) Framework” and show how it can be adopted to model, manage, access and exchange hierarchically structured resources. The presentation of this framework also shows how it can be used to address interoperability in Digital Libraries. As a guide use case we make use of archives both because of their peculiar characteristics and because of the challenges we have to accomplish to model, manage, access, and exchange their resources. The results that are here presented are rooted on those presented in [11].

The presentation is organized as follows: Section 2 presents the objectives and the key contributions of this chapter and introduces the composition of the NESTOR Framework which is defined by the NESTOR Model, the NESTOR Algebra and the NESTOR Prototype. Section 3 describes the context background of the work; it introduces the main concepts about archives and related technologies as well as a panoramic on the Digital Library technologies we are going to employ in this chapter. Section 4 describes the NESTOR Model based on two set data models and Section 5 depicts the main features of the NESTOR Algebra. In Section 6 we show how the NESTOR Prototype can be modeled and used to achieve the presented requirements. Lastly, we draw some conclusions and give indications for future work in Section 7.

2 Objectives and Contributions

The main target of this chapter is to provide a framework allowing us to model, manage, access and exchange hierarchically structured resources. A key aspect is to envision this framework in the context of Digital Libraries and to use it as a mean to address the complex and multiform concept of interoperability when dealing with hierarchical structures. A significant goal is to understand which is the best option for modeling hierarchies in order to meet the higher number of user and system requirements. The modeling and representation of data (or more in general resources) is a fundamental step towards building automatic and effective systems and providing services and functionalities to the users. In this context we take into account one of the most important data structures in computer science – the tree data structure [19]; this is widely adopted in many scientific fields to model and represent hierarchies. The tree data structure is often regarded as the only way to model hierarchies; our aim is to investigate alternative data models that can do the work and then compare their effectiveness in specific application contexts.

We propose the NESTOR Framework as a conceptual and logical mean for modeling and representing hierarchically structured data and specifically to overcome some of the difficulties that we encounter when we have to address interoperability in Digital Libraries. There is the lack of a general framework satisfying these requirements which often need to be addressed on a case-by-case basis. The NESTOR

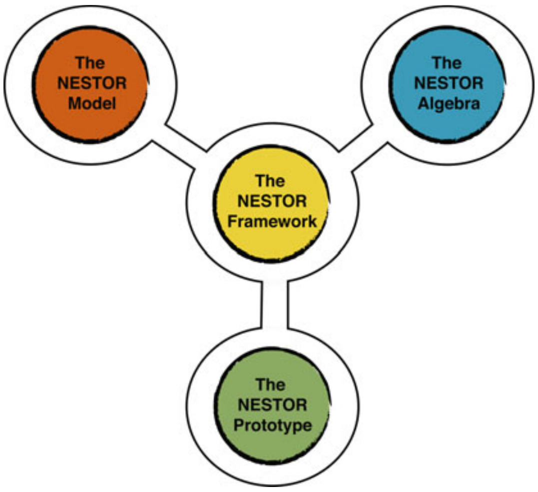


Fig. 1 The graphical outline of the composition of the NESTOR Framework: Model, Algebra and Prototype

Framework is constituted by three parts – pointed out in Figure 1: The *NESTOR Model*, the *NESTOR Algebra* and the *NESTOR Prototype*.

The NESTOR Model is the heart of the Framework; it is based on two set data models called *Nested Set Model* (NS-M) and *Inverse Nested Set Model* (INS-M) which are based on an organization of nested sets. The foundational idea underlying these set data models is that an opportune set organization can maintain all the features of a tree data structure with the addition of some new relevant functionalities. We define these functionalities in terms of flexibility of the model, rapid selection and isolation of easily specified subsets of data and extraction of only those data necessary for satisfying specific needs. We can use these set data models to represent hierarchical structures disclosing a variety of properties which can be related to the properties of the tree data structure and which are also peculiar of these models. The representation of hierarchies by one of these models lays the ground for an environment leading to new ways of modeling and consequently accessing, managing and querying hierarchical data.

Each data model has to specify a set of operations to manipulate and query the data represented by a specific model; for instance, in the relational model this operation set is defined by the relational algebra [6]. A formal bulk algebra is essential to a data model first of all because it provides a formal basis for the operations on the data sets and second of all because it is used as a basis to implement and optimize the queries written in some query language against these data sets. We develop an algebra, called the NESTOR Algebra, for the manipulation and query of data represented throughout the set data models defined in the NESTOR Model.

The NESTOR Prototype gives an actual instantiation of the model and of the algebra allowing the application of the formal concepts defined in the NESTOR Model and Algebra. The prototype is presented by the use case of the archives describing how a hierarchy can be modeled by means of the NESTOR Model and specifically how the archival records can be represented through it. The NESTOR Prototype takes into account the dimensions of interoperability in Digital Libraries [13] showing how the adoption of the NESTOR Model and the exploitation of the Algebra addresses several interoperability issues. In particular, we face the problem of access and exchange of hierarchically organized resources in distributed environment and discuss the relationships between the NESTOR Prototype and Digital Library technologies such as the OAI-PMH protocol. Furthermore, we illustrate how we can address multilingualism in the archival context by exploiting widely-adopted techniques and technologies.

3 The Background Context and Technologies

The background of this chapter relies on the archives environment, thus it is fundamental to describe the nature of archival practice and the peculiarities of archival resources. In the beginning we present the nature of archives and then we talk about digital archives. In this context we present the standard XML metadata format adopted by the archives, i.e. is the Encoded Archival Description (EAD).

Two important Digital Library technologies are the OAI-PMH protocol and the Dublin Core. We will exploit these technologies in the context of the NESTOR Prototype, thus it is worthwhile to describe their characteristics and functionalities.

3.1 Archives and Archival Descriptions

An archive is not simply constituted by a series of objects that have been accumulated and filed with the passing of time. Instead, it represents the trace of the activities of a physical or juridical person in the course of their business which is preserved because of their continued value. Archives have to keep the context in which their records[4] have been created and the network of relationships between them in order to preserve their informative content and provide understandable and useful information over time.

Archival description is defined in [23] as “the process of analyzing, organizing, and recording details about the formal elements of a record or collection of records, to facilitate the work’s identification, management, and understanding”; archival descriptions have to reflect the peculiarities of the archive, retain all the informative power of a record, and keep trace of the provenance and original order in which resources have been collected and filed by archival institutions [12]. This is emphasized by the central concept of *fonds*[5], which should be viewed primarily as an

[4] In [21] a record is defined as: “Any document made or received and set aside in the course of a practical activity”.

[5] The term *fonds* is not a commonly used English word but it is derived from the French and it is used both for the singular and plural form of the noun.

"intellectual construct", the conceptual "whole" that reflects an organic process in which a records creator produces or accumulates series of records [8]. In this context, provenance becomes a fundamental principle of archives; the principle of the "*respect des fonds*" which dictates that resources of different origins be kept separate to preserve their context; the "*respect des fonds*" is often regarded as the principle of *provenance* [12, 10].

[10] highlights that maintaining provenance leads archivists to evaluate records on the basis of the importance of the creator's mandate and functions, and fosters the use of a hierarchical method. The hierarchical structure of the archive expresses the relationships and dependency links between the records of the archive by using what is called the archival bond defined as "the interrelationships between a record and other records resulting from the same activity" [23]. Archival bonds, and thus relations, are constitutive parts of an archival record: if a record is taken out from its context and has lost its relations, its informative power would also be considerably affected. Therefore, archival descriptions need to be able to express and maintain such structure and relationships in order to preserve the context of a record. To this end, the International Council on Archives (ICA)[6] has developed a general standard for archival description called International Standard for Archival Description (General) (ISAD(G)) [15]. According to ISAD(G), archival description proceeds from general to specific as a consequence of the provenance principle and has to show, for every unit of description, its relationships and links with other units and to the general fonds. Therefore, archival descriptions produced according to the ISAD(G) standard take the form of a tree which represents the relationships between more general and more specific archive units going from the root to the leaves of the tree.

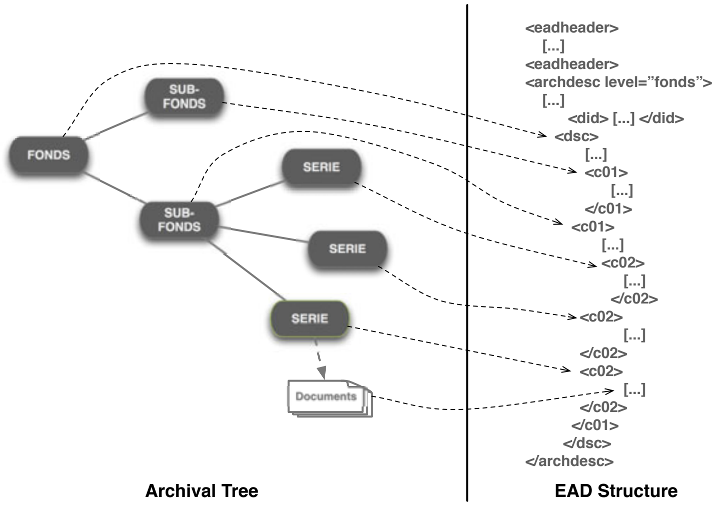
3.2 EAD: Encoded Archival Description

EAD is an archival description metadata standard that reflects and emphasizes the hierarchical nature of ISAD(G) [24]. EAD fully enables the expression of multiple description levels central to most archival descriptions and reflects hierarchy levels present in the resources being described. EAD cannot be considered a one-to-one ISAD(G) implementation, although it does respect ISAD(G) principles and is useful for representing archival hierarchical structures. EAD is composed of three high-level components: `<eadheader>`, `<frontmatter>`, and `<archdesc>`.

The `<eadheader>` contains metadata about the archive descriptions and includes information about them such as title, author, and date of creation. The `<frontmatter>` supplies publishing information and is an optional element, while the `<archdesc>` contains the archival description itself and constitutes the core of EAD. The `<archdesc>` may include many high-level sub-elements, most of which are repeatable. The most important element is the `<did>` or descriptive identification which describes the collection as a whole. The `<did>` element is composed of numerous sub-elements that are intended for brief, clearly designated

[6] `http://www.ica.org/`

statements of information and they are available at every level of description. Finally, the <archdesc> contains an element that facilitates a detailed analysis of the components of a fonds, the <dsc> or description subordinate components. The <dsc> contains a repeatable recursive element, called <c> or component. A component may be an easily recognizable archival entity such as series, subseries or items. Components not only are nested under the <archdesc> element, they usually are nested inside one another. Components usually are indicated with <cN> tag, where $N \in \{01, 02, \ldots, 12\}$.



```
<eadheader>
   [...]
<eadheader>
<archdesc level="fonds">
   [...]
      <did> [...] </did>
      <dsc>
         [...]
         <c01>
            [...]
         </c01>
       <c01>
          [...]
          <c02>
             [...]
          </c02>
       <c02>
          [...]
       </c02>
       <c02>
          [...]
       </c02>
      </c01>
     </dsc>
</archdesc>
```

Fig. 2 How an archive represented as a tree is mapped into an EAD XML file

EAD reflects the archival structure and holds relations between entities in an archive. In addition, EAD encourages archivists to use collective and multilevel description, and because of its flexible structure and broad applicability, it has been embraced by many repositories [17].

On the other hand, EAD allows for several degrees of freedom in tagging practice, which may turn out to be problematic in the automatic processing of EAD files, since it is difficult to know in advance how an institution will use the hierarchical elements. The EAD permissive data model may undermine the very interoperability it is intended to foster. Indeed, it has been underlined that only EAD files meeting stringent best practice guidelines are shareable and searchable [26]. Moreover, there is also a second relevant problem related to the level of material being described. Unfortunately, the EAD schema rarely requires a standardized description of the level of the materials being described, since the <level> attribute is required only in the <archdesc> tag, while it is optional in <cN> components and in very few EAD files this possibility is used, as pointed out by [25]. As a consequence, the level of description of the lower components in the hierarchy needs to be inferred by navigating the upper components, maybe up to the `<archdesc>`, where the presence of the `<level>` attribute is mandatory. Therefore, access to individual items might be difficult without taking into consideration the whole hierarchy.

We highlight this fact in Figure 2 where we present the structure of an EAD file. In this example we can see the top-level components `<eadheader>` and `<archdesc>` and the hierarchical part represented by the `<dsc>` component; the `<level>` attribute is specified only in the `<archdesc>` component. Therefore, the archival levels described by the components of the `<dsc>` can be inferred only by navigating the whole hierarchy.

3.3 OAI-PMH and Dublin Core

OAI-PMH is based on the distinction between two main components that are Data Provider and Service Provider. Data Providers are repositories that export records in response to requests from a software service called harvester. On the other hand, Service Providers are those services that harvest records form Data Providers and provide services built on top of aggregated harvest metadata.

The protocol defines two kinds of harvesting procedures: incremental and selective harvesting. Incremental harvesting permits users to query a Data Provider and ask it to return just the new, changed or deleted records from a certain date or between two dates. Selective harvesting is based on the concept of *OAI set*, which enables logical data partitioning by defining groups of records. Selective harvesting is the procedure that permits the harvesting only of metadata owned by a specified OAI set. [30] states that in OAI-PMH a set is defined by three components: `setSpec` which is mandatory and a unique identifier for the set within the repository, `setName` which is a mandatory short human-readable string naming the set, and `setDesc` which may hold community-specific XML-encoded data about the set.

OAI set organization may be flat or hierarchical, where hierarchy is expressed in `setSpec` field by the use of a colon [:] separated list indicating the path from the root of the set hierarchy to the respective node. For example if we define an OAI set for whose `setSpec` is *"A"*, its sub-set "B" would have *"A:B"* as `setSpec`. In this case "B" is a proper sub-set of "A": $B \subset A$. When a repository defines a set organization it must include set membership information in the headers of items returned to the harvester requests. Harvesting from a set which has sub-sets will cause the repository to return metadata in the specified set and recursively to return metadata from all the sub-sets. In our example, if we harvest set A, we also obtain the items in sub-set B [29].

The Dublin Core (DC) metadata format is tiny, easy-to-move, shareable and remarkably suitable for a distributed environment. Thanks to these characteristics it is required as the lowest common denominator in OAI-PMH. Thus, DC metadata are very useful in information sharing but are not broadly used by archivists. Indeed, the use of DC seems to flatten out archive structure and lose context and hierarchy information. For this reason, even though DC is used in several contexts ranging from Web to digital libraries, it is less used in the archival domain. Nevertheless, we can apply it to the archival domain and meet the three requirements discussed above, if we use it in combination with OAI-PMH: in this way, the OAI set provides us with context and hierarchy requirements compliance, while the DC metadata format gives us the expected variable granularity support.

4 The NESTOR Model

The NESTOR Framework is the composition of three main parts: the Model, the Algebra and the Prototype. The NESTOR Model is the core of the framework because it defines the set data models on which every component of the framework relies. In Figure 3 we can see a graphical representation of the principal components composing the NESTOR Model. In the upper part we have the set data models – i.e. the Nested Set Model (NS-M) and the Inverse Nested Set Model (INS-M). The second component represents the properties of the set data models such as: the mapping function to go from the NS-M to the INS-M and vice versa, the meaning of the union or intersection of two sets or the definition of distance measures. The latter component represents the relationships between the set data models and the tree data structure; this component contains the functions for mapping a tree into one of the two models and it compares the properties of the tree with the properties of the set data models.

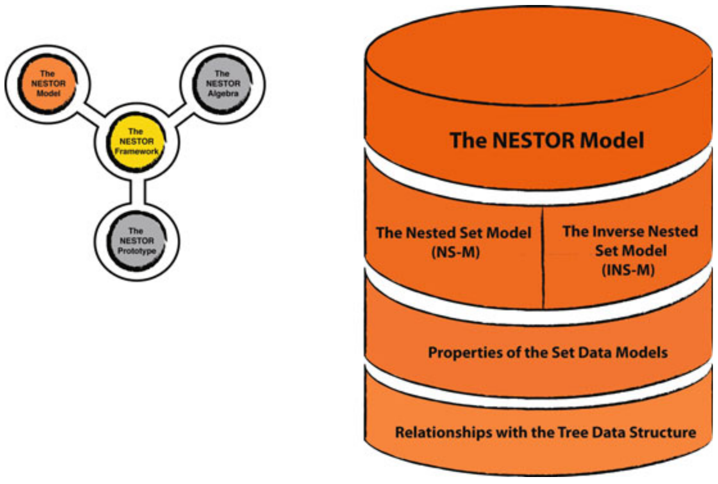

The NESTOR Model
The NESTOR Algebra
The NESTOR Framework
The NESTOR Prototype
The NESTOR Model
The Nested Set Model (NS-M)
The Inverse Nested Set Model (INS-M)
Properties of the Set Data Models
Relationships with the Tree Data Structure

Fig. 3 The main components of the NESTOR Model

The formal definition of the NESTOR Model within all its components relies on set theory, and particularly, on the basic concept of *family of subsets*. It is not in the scope of this chapter to give a complete mathematical definition of all the components of the NESTOR Model. For this reason, in order to properly understand how the set data models are defined it is worthwhile to get an intuitive idea of their main characteristics. After this intuitive presentation we explain some basic concepts of set theory which allow us to understand the formal definition of the models and to introduce the minimum set of notation and terminology indispensable to understand the properties of the models, the relationships with the tree data structure and, afterwards, the NESTOR Algebra.

Now, we informally present the two data models with examples of mapping between them and a sample tree, with a clear understanding that these models are independent from the tree data structure. The first model we present is the **Nested Set Model** (NS-M). The intuitive graphic representation of a tree as an organization of nested sets was used in [19] to show different ways of representing tree data structure and in [5] to explain an alternative way for solving recursive queries over trees in SQL language. An organization of sets in the NS-M is a collection of sets in which any pair of sets is either disjoint or one contains the other. In Figure 4 (b) we can see how a sample tree is mapped into an organization of nested sets based on the NS-M.

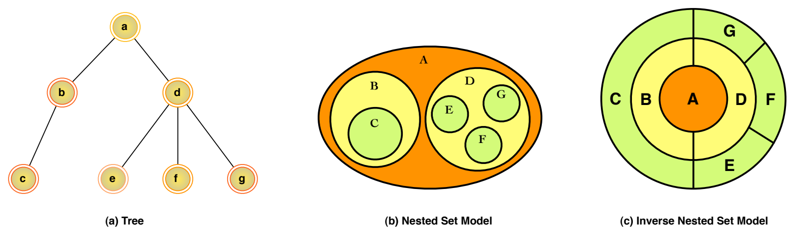


Fig. 4 (a) A tree. (b) Euler-Venn Diagram of a NS-M. (c) Doc-Ball representation of a INS-M.

From Figure 4 (b) we can see that each node of the tree is mapped into a set, where child nodes become *proper subsets* of the set created from the parent node. Every set is subset of at least of one set; the set corresponding to the tree root is the only set without any supersets and every set in the hierarchy is subset of the root set. The external nodes are sets with no subsets. The tree structure is maintained thanks to the nested organization and the relationships between the sets are expressed by the set inclusion order. Even the disjunction between two sets brings information; indeed, the disjunction of two sets means that these belong to two different branches of the same tree.

The second data model is the **Inverse Nested Set Model** (INS-M). We can say that a tree is mapped into the INS-M by transforming each node into a set, where each parent node becomes a subset of the sets created from its children. The set created from the tree's root is the only set with no subsets and the root set is a proper subset of all the sets in the hierarchy. The leaves are the sets with no supersets and they are sets containing all the sets created from the nodes composing the tree path from a leaf to the root. An important aspect of INS-M is that the intersection of every couple of sets obtained from two nodes is always a set representing a node in the tree. The intersection of all the sets in the INS-M is the set mapped from the root of the tree.

Unlike the NS-M, representing the INS-M with the Euler-Venn diagrams is not very expressive and can be confusing for the reader [1]. We can represent the INS-M in a straightforward way by means of the "*DocBall representation*" [9]. The DocBall representation is used in [9] to depict the structural components of the documents and can be considered as the representation of a tree structure. We exploit the DocBall ability to show the structure of an object and to represent the "*inclusion order of one or more elements in another one*" [31]. The DocBall is composed of a set of circular sectors arranged in concentric rings as shown in Figure 4 (c). In a DocBall each ring represents a level of the hierarchy with the center (level 0) representing the root. In a ring, the circular sectors represent the nodes in the corresponding level. We use the DocBall to represent the INS-M, thus for us each circular sector corresponds to a set.

In Figure 4 (c) we can see the INS-M mapping of a sample tree by means of the DocBall representation. The root "*a*" of the tree is mapped into set "*A*" represented by the inner ring at level 0 of the DocBall; at level 1 we find the children of the root and so on. With this representation a subset is presented in a ring within the set including it. Indeed, we can see that set A is included in all the other sets. If the intersection of two or more sets is empty then these sets have no common circular sector in the inner rings of the DocBall; in the INS-M this is not possible because the set representing the root (A) is common to all the sets in the INS-M. For instance, we can see that the circular sectors C and E have in common only A, indeed $C \cap E = A$; instead, G and E have in common sectors D and A, thus $G \cap E = \{D, A\}$.

Both the NS-M and the INS-M have been presented as "organizations of nested sets"; two sets are nested if one contains the other and thus, if one is the subset of the other one. The nesting between two sets determines an order inclusion between them. Let us consider a set, call it A, that contains all the elements organized in the hierarchy we want to represent throughout the NS-M or the INS-M. Now, let us consider two sets, call them A_1 and A_2, which are subsets of A such that: $A_1 \subset A$ and $A_2 \subset A$. The collection $\mathscr{C}$ composed by the two sets A_1, A_2 is called the collection of subsets of A; a family of subsets of A is just the collection $\mathscr{C}$ indexed by an "index set". The following definition formally states the very concepts we have just described.

Definition 1. *Let A be a set, I a non-empty set and $\mathscr{C}$ a collection of subsets of A. Then a bijective function $\mathscr{A} : I \longrightarrow \mathscr{C}$ is a* ***family*** *of subsets of A. We call I the* ***index*** *set and we say that the collection $\mathscr{C}$ is* ***indexed*** *by I.*

We use the extended notation $\{A_i\}_{i \in I}$ to indicate the family $\mathscr{A}$; the notation $A_i \in \{A_i\}_{i \in I}$ means that $\exists\ i \in I \mid \mathscr{A}(i) = A_i$. In the rest of the chapter we will use the shorthand notation $\mathscr{A}$ when there is no risk of ambiguity and when it is not necessary to indicate the index set. We call **subfamily** of $\{A_i\}_{i \in I}$ the **restriction** of $\mathscr{A}$ to $J \subseteq I$ and we denote this with $\{B_j\}_{j \in J} \subseteq \{A_i\}_{i \in I}$.

From this definition we can see that an organization of nested sets in set theory is defined as a family of subsets or just "family" if in the context in which it is used there is no risk of ambiguity. Thus, in the context of the NS-M we have Nested Set Families (NS-F) and in INS-M we have Inverse Nested Set Families (INS-F). The differences between these two models are expressed in the constraints we impose respectively on a NS-F and a INS-F. Let us consider the formal definition of NS-F.

Definition 2. *Let A be a set and let $\{A_i\}_{i\in I}$ be a family. Then $\{A_i\}_{i\in I}$ is a* ***Nested Set Family*** *if:*

$$A \in \{A_i\}_{i\in I}, \tag{1}$$

$$\emptyset \notin \{A_i\}_{i\in I}, \tag{2}$$

$$\forall A_h, A_k \in \{A_i\}_{i\in I}, h \neq k \mid A_h \cap A_k \neq \emptyset \Rightarrow A_h \subset A_k \vee A_k \subset A_h. \tag{3}$$

Thus, we define a Nested Set Family (NS-F) as a family where three conditions must hold. The first condition (1) states that set A which contains all the sets in the family must belong to the NS-F. The second condition states that the empty-set does not belong to the NS-F and the last condition (3) states that the intersection of every couple of distinct sets in the NS-F is not the empty-set only if one set is a proper subset of the other one [14, 3].

In the same way we define the Inverse Nested Set Model (INS-M):

Definition 3. *Let A be a set and let $\{A_i\}_{i\in I}$ be a family. Then $\{A_i\}_{i\in I}$ is an* ***Inverse Nested Set Family*** *if:*

$$\emptyset \notin \{A_i\}_{i\in I}, \tag{4}$$

$$\forall \{B_j\}_{j\in J} \subseteq \{A_i\}_{i\in I} \Rightarrow \bigcap_{j\in J} B_j \in \{A_i\}_{i\in I}. \tag{5}$$

$$\begin{aligned} &\forall \{B_j\}_{j\in J} \subseteq \{A_i\}_{i\in I} \\ &\Rightarrow \exists B_k \in \{B_j\}_{j\in J} \mid \forall B_h \in \{B_j\}_{j\in J}, B_h \subseteq B_k \\ &\Rightarrow \forall B_h, B_g \in \{B_j\}_{j\in J}, B_h \subseteq B_g \vee B_g \subseteq B_h. \end{aligned} \tag{6}$$

Thus, we define an Inverse Nested Set Family (INS-F) as a family where three conditions must hold. The first condition (4) states that the empty-set does not belong to the INS-F. The second condition (5) states that the intersection of every subfamily of the INS-F belongs to the INS-F itself. Condition 6 states that for every possible subfamily of a INS-F there cannot exist a set in the subfamily which is a superset of all the other sets in the subfamily, unless all the sets in the subfamily form a *chain*[7].

In a family of subsets the sets establish a hierarchical relationship one with the other as well as in the tree data structure the nodes are in a parent-child or ancestor-descendant relationship. Also in a family we can have different kind of relationships between the sets; let us consider a family $\{A_i\}_{i\in I}$ where $A_1, A_2 \in \{A_i\}_{i\in I}$ are two

[7] A family of subsets $\{A_i\}_{i\in I}$ forms a chain (or it is linearly ordered) if each set in $\{A_i\}_{i\in I}$ is nested inside the next: the family $\{A_i\}_{i\in I}$ is defined a **chain** if $\forall A_j, A_k \in \{A_i\}_{i\in I}, A_j \subseteq A_k \vee A_k \subseteq A_j$.

sets, A_2 is a direct subset of the set A_1 if and only if it does not exists a third set $A_3 \in \{A_i\}_{i \in I}$ such that $A_2 \subset A_3 \subset A_1$. In the same way we say that A_1 is a direct superset of A_2 if and only if it does not exists a third set $A_3 \in \{A_i\}_{i \in I}$ such that $A_2 \subset A_3 \subset A_1$. The following definition presents the concept of collection of proper subsets and supersets; afterwards we present the definition of collection of proper direct subsets and supersets.

Definition 4. *Let $\{A_i\}_{i \in I}$ be a family and $A_j \in \{A_i\}_{i \in I}$ be a set. We define $\mathscr{S}^+_{\mathscr{A}}(A_j) = \{A_k : A_k \in \{A_i\}_{i \in I} \wedge A_k \subset A_j\}$ to be the* ***collection of proper subsets*** *of A_j in the family $\mathscr{A}$. In the same way we define the* ***collection of proper supersets*** *of A_j in the family $\mathscr{A}$ as $\mathscr{S}^-_{\mathscr{A}}(A_j) = \{A_k : A_k \in \{A_i\}_{i \in I} \wedge A_j \subset A_k\}$.*

It is worthwhile for the rest of the work to introduce the definition of *collection of direct super/sub-sets* as a restriction of the above defined collection of proper super/sub-sets.

Definition 5. *Let $\{A_i\}_{i \in I}$ be a family and $A_j \in \{A_i\}_{i \in I}$ be a set. We define $\mathscr{D}^+_{\mathscr{A}}(A_j) = \{A_k : A_k \in \{A_i\}_{i \in I} \wedge A_k \subset A_j \wedge \nexists A_t \in \{A_i\}_{i \in I} \mid A_k \subset A_t \subset A_j\}$ to be the* ***collection of direct subsets*** *of A_j in the family $\mathscr{A}$. In the same way we define the* ***collection of direct supersets*** *of A_j in the family $\mathscr{A}$ as $\mathscr{D}^-_{\mathscr{A}}(A_j) = \{A_k : A_k \in \{A_i\}_{i \in I} \wedge A_j \subset A_k \wedge \nexists A_t \in \{A_i\}_{i \in I} \mid A_j \subset A_t \subset A_k\}$.*

Now that we have at our disposal the fundamental concepts of set theory necessary for understanding the work and that we have formally defined the NS-M and the INS-M, we can examine the relationships between them and their properties. From the collection of properties of these models we choose to introduce only those used in this chapter to show how the NESTOR Framework addresses the interoperability issues we have presented. As we have done for the definition of the models, we present some of their properties and their relationships with the tree data structure in an informal way by means of some examples. In [11, 2] the reader can find the formal definitions and theorems proving the claims that we present in the following. In order to explain the characteristics of the set data models we have already shown how a tree can be mapped into a NS-F or an INS-F; in the same way it is important to show how a NS-F can be mapped into a INS-F and vice versa, thus establishing a bijective relation between the set data models.

Example 1. *Let $\{A_i\}_{i \in I}$ be a NS-F and let $\{A_i\}_{i \in I} = \{A_1, A_2, A_3, A_4, A_5\}$ where $A_1 = \{a,b,c,d,e,f,g\}$, $A_2 = \{b,g\}$, $A_3 = \{c,d,e\}$, $A_4 = \{d\}$ and $A_5 = \{e\}$. Then we can map the NS-F $\{A_i\}_{i \in I})$ into a correspondent INS-F $\{B_j\}_{j \in J} = \{B_1, B_2, B_3, B_4, B_5\}$, mapping each set of $\{A_i\}_{i \in I})$ into a set of $\{B_j\}_{j \in J})$:*

$B_1 = \bigcup_{A_t \in \{A_1 \cup \mathscr{S}^-_{\mathscr{A}}(A_1)\}} (A_t \setminus \bigcup \mathscr{S}^+_{\mathscr{A}}(A_t)) =$
$A_1 \setminus \bigcup \{A_2, A_3, A_4, A_5\} = \{a,b,c,d,e,f,g\} \setminus \{b,c,d,e,g\} = \{a,f\}$.

$B_2 = \bigcup_{A_t \in \{A_2 \cup \mathscr{S}^-_{\mathscr{A}}(A_2)\}} (A_t \setminus \bigcup \mathscr{S}^+_{\mathscr{A}}(A_t)) =$
$A_2 \setminus \{\emptyset\} \cup A_1 \setminus \bigcup \{A_2, A_3, A_4, A_5\} = \{b,g\} \cup \{a,f\} = \{a,f,b,g\}$.

$B_3 = \bigcup_{A_t \in \{A_3 \cup \mathscr{S}^-_{\mathscr{A}}(A_3)\}} (A_t \setminus \bigcup \mathscr{S}^+_{\mathscr{A}}(A_t)) =$
$A_3 \setminus \{A_4, A_5\} \cup A_1 \setminus \bigcup \{A_2, A_3, A_4, A_5\} = \{c\} \cup \{a,f\} = \{c,a,f\}$.

$B_4 = \bigcup_{A_t \in \{A_4 \cup \mathscr{S}^-_{\mathscr{A}}(A_4)\}} (A_t \setminus \bigcup \mathscr{S}^+_{\mathscr{A}}(A_t)) =$
$A_4 \setminus \{\emptyset\} \cup A_3 \setminus \{A_4, A_5\} \cup A_1 \setminus \bigcup \{A_2, A_3, A_4, A_5\} = \{d\} \cup \{c\} \cup \{a, f\} = \{d, c, a, f\}.$

$B_5 = \bigcup_{A_t \in \{A_5 \cup \mathscr{S}^-_{\mathscr{A}}(A_5)\}} (A_t \setminus \bigcup \mathscr{S}^+_{\mathscr{A}}(A_t)) =$
$A_5 \setminus \{\emptyset\} \cup A_3 \setminus \{A_4, A_5\} \cup A_1 \setminus \bigcup \{A_2, A_3, A_4, A_5\} = \{e\} \cup \{c\} \cup \{a, f\} = \{e, c, a, f\}.$

In Figure 5 we can see a graphical representation of the NS-F mapped into a INS-F. The mapping between $\{A_i\}_{i \in I}$) and $\{B_j\}_{j \in J}$ can be defined by means of a function; we define $\zeta : \{A_i\}_{i \in I} \to \{B_j\}_{j \in J}$ to be a function such that $\forall A_k \in \{A_i\}_{i \in I}$, $\exists B_k \in \{B_j\}_{j \in J} \mid B_k = \bigcup_{A_t \in \{A_k \cup \mathscr{S}^-_{\mathscr{A}}(A_k)\}} (A_t \setminus \bigcup \mathscr{S}^+_{\mathscr{A}}(A_t))$.

In the same way we can define the function $\xi : \{B_j\}_{j \in J} \to \{A_i\}_{i \in I}$ such that $\forall B_k \in \{B_j\}_{j \in J}, \exists A_k \in \{A_i\}_{i \in I} \mid A_k = \bigcup (B_k \cup \mathscr{S}^-_{\mathscr{B}}(B_k)) \setminus \bigcup \mathscr{S}^+_{\mathscr{B}}(B_k)$. The function ξ maps $\{B_j\}_{j \in J}$ into $\{A_i\}_{i \in I}$ thus a NS-F into a INS-F.

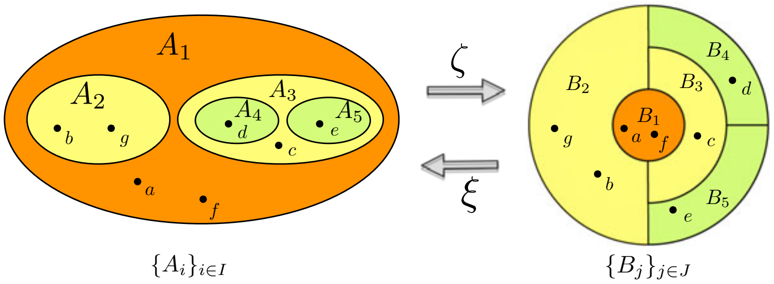


Fig. 5 A NS-F $\{A_i\}_{i \in I}$ and its correspondent INS-F $\{B_j\}_{j \in J}$

This example showed us that if we model a hierarchy by means of a model we can map it into the other model and go from one to the other whenever it is necessary. We will see that this possibility is very useful in a concrete application because it allows us to be free to change the representation of a hierarchy and thus not to be bound to a single representation. The possibility of going from one model to the other is particularly useful when we have to exploit a property of a specific set data model; as an example we present the problem of how to find the **lowest common ancestor** in a tree $T(V, E)$ where V is the set of vertexes and E is the set of edges of the tree. We already know that $T(V, E)$ can be mapped in a NS-F or in a INS-F; let us see how the concept of lowest common ancestor is handled in the set data models. We can say that the lowest common ancestor, call it $v_t \in V$, of nodes $v_j \in V$ and $v_k \in V$ in a tree $T(V, E)$ is the ancestor of v_j and v_k that is located farthest from the root. Many algorithms have been proposed in the literature [4] to efficiently determine the lowest common ancestor of a tree. The same operation can be done both in the NS-M and the INS-M. If we map the tree $T(V, E)$ in a NS-F $\{A_i\}_{i \in I}$ each node $v_i, v_k, v_t \in V$ is mapped in a correspondent set $A_i, A_k, A_t \in \{A_i\}_{i \in I}$; the same operation can be done by mapping the same tree into an INS-F $\{B_j\}_{j \in J}$ where each node $v_i, v_k, v_t \in V$ is mapped in a correspondent set $B_i, B_k, B_t \in \{B_j\}_{j \in J}$.

In $\{A_i\}_{i \in I}$ set A_t represents the lowest common ancestor between the sets A_i and A_k. In order to determine A_t we have to consider a collection $\mathscr{C}_i = A_i \cup \mathscr{S}^+_{\mathscr{A}}(A_i)$

containing A_i and all its supersets in the family and a collection $\mathscr{C}_k = A_k \cup \mathscr{S}^{+}_{\mathscr{A}}(A_k)$ containing A_k and all its supersets in the family; then, we have to intersect these two collections $C_x = C_i \cap C_k$. The collection C_x contains all the sets which are common supersets of A_j and A_k and the set A_t is the set with smaller cardinality in C_x. So, to determine the correspondent of the lowest common ancestor $v_t \in V$ in the NS-F $\{A_i\}_{i \in I}$ we have to do several operations: determine all the supersets of A_i and A_k, intersect the collections containing these sets, calculate the cardinality of all the sets in the intersection and take the set with the smaller one.

On the other hand, in $\{B_j\}_{j \in J}$ set B_t represents the lowest common ancestor between sets B_i and B_k. In order to determine B_t we have to intersect B_i and B_k and the resulting set is the correspondent of the lower common ancestor. So, in INS-M the problem of finding the lowest common ancestor is reduced to a single intersection between two sets; we can see that the choice of the set data model to adopt to represent a hierarchy is going to influence the efficiency with which we can do some operations. The best choice between one model or the other depends on the application environment we are considering; at the same time, the possibility to go from a model to the other on the fly allows us to choose the best option on a case-by-case basis.

5 The NESTOR Algebra

We designed an algebra, called the NESTOR Algebra, for the manipulation of data represented throughout the set data models defined in the NESTOR Model. In order to clarify the functioning and the fundamental principles on which the NESTOR Algebra is based we will relate them to the widely known relational algebra basics. Most of the relational operations such as selection, projection, product and set operations are important operations we want to perform on the data represented as families of sets. The NESTOR Algebra can be uniformly adopted by the NS-F and the INS-F. In Figure 6 we can see a graphical representation of the basic components composing the NESTOR Algebra.

The first component contains the data model on which the algebra is based, the predicates of the algebra and the fundamental concept of pattern family on which the whole algebra relies. The second component contains the definition of the operators of the algebra which are both manipulation and query ones: renaming, value update, insertion, deletion, selection, projection, union, intersection, set difference, product, join, grouping and aggregation. The third component contains the relationships of the NESTOR Algebra with the relational algebra and the Tree Algebra for XML [16, 22]. In this chapter we do not describe each one of the algebra operators and characteristics, nor do we show its completeness for the relational algebra and the Tree Algebra for XML [16]; we present the main features of the algebra in order to understand its functioning and its possible uses to address the interoperability issues we have presented.

A central feature of the relational algebra is the declarative expression of queries over the collection of tuples; an important thing we have to take into consideration is

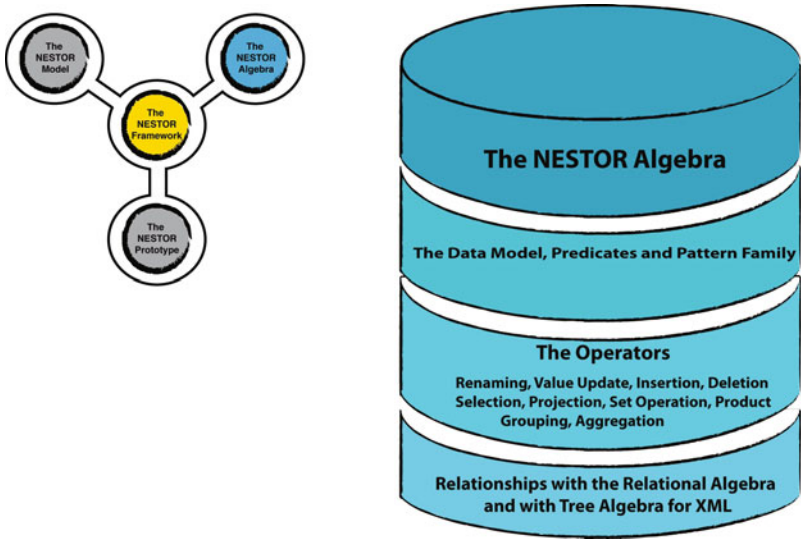

The NESTOR Model
The NESTOR Algebra
The NESTOR Framework
The NESTOR Prototype
The NESTOR Algebra
The Data Model, Predicates and Pattern Family
The Operators
Renaming, Value Update, Insertion, Deletion
Selection, Projection, Set Operation, Product
Grouping, Aggregation
Relationships with the Relational Algebra and with Tree Algebra for XML

Fig. 6 The main components of the NESTOR Algebra

that the manipulation of data in the NESTOR Framework often uses structural constructs, and element inclusion (i.e. the determination of set/superset relationships between two sets). We need to maintain this information when we manipulate the sets organized into families in the set data models. Thus, in the NESTOR Algebra a whole family of subsets is the fundamental unit, similar to a relational tuple. We do not present an algebra where the families are transformed in a collection of tuples to be processed and then re-transformed in families following a relational construction/deconstruction paradigm as is done in XML shredding [27]. Instead, we follow the approach adopted in Tree Algebra for XML [16] developed for the manipulation of XML data modeled as forests of labeled, ordered trees; thus, we choose to manage collections of families of sets directly.

A relation in the relational model is a collection of tuples and the counterpart in the nested sets models is a collection of families. Each relational algebra operator takes one or more relations as input and produces a relation as output. Correspondingly, each operator of the NESTOR Algebra takes a collection of one or more families as input and produces a collection of one or more families as output. This means that the NESTOR Algebra considers a whole family as the basic unit and not, for instance, a single set belonging to a family or the elements belonging to the sets in a family.

Predicates are central to much of querying. While the choice of the specific set of allowable predicates is orthogonal to the NESTOR Algebra, any given implementation will have to make a choice in this matter. In the NESTOR Algebra we point out structural and content predicates; the structural predicates allow us to express conditions on the structure of the families of subsets, instead the content predicates allow us to define conditions on the elements belonging to the sets. For instance, we use a structural predicate if we want to express a condition saying that a set $A_j \in \mathscr{A}_I$[8] must be a subset of another set $A_k \in \mathscr{A}_I$. On the other hand, we use a content predicate when we need to express conditions on specific elements belonging to the sets of interest.

The following example presents the NS-F we will use as a basis in the description of the algebra; it is a toy representation of an archive where the label of every element is a simple string that also give an indication of the archival information brought by that element.

Example 2. *Let* $\{B_j\}_{j \in J}$ *be a NS-F where* $\{B_j\}_{j \in J} = \{B_1, B_2, B_3, B_4, B_5, B_6\}$.

$B_1 = \{$ `summary`, `biography`, `chronology`, `programA`, `letterG`, `letterF`, `programC`, `programD`, `letterA`, `letterB`, `letterC`, `testamentA`, `letterD`, `testamentB`, `testamentC`$\}$, $B_2 = \{$`programA`, `letterG`, `letterF`$\}$.

$B_3 = \{$`programC`, `programD`$\}$, $B_4 = \{$`letterD`, `testamentB`, `testamentC`$\}$, $B_5 = \{$`letterA`, `letterB`$\}$ *and* $B_6 = \{$`letterC`, `testamentA`$\}$.

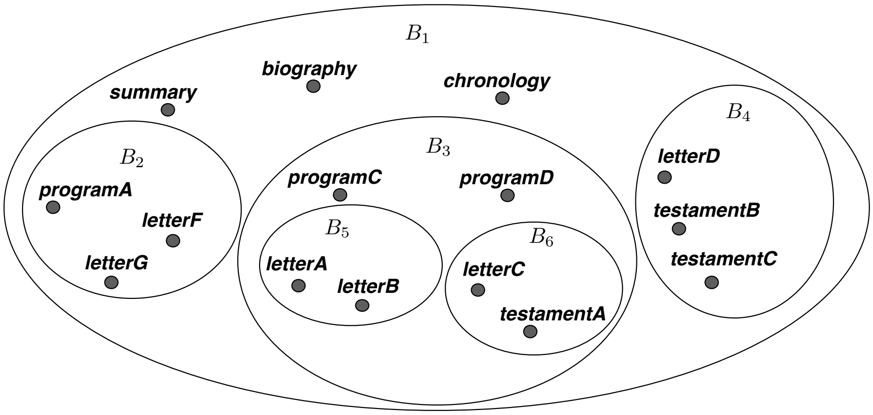


Fig. 7 Venn-diagram of the NS-F described in Example 2

A basic syntactic requirement of any algebra is the ability to specify the attributes of interest. In relational algebra this is accomplished straightforwardly by considering domain-ordered relations or using names instead of position numbers for "identifying the domains" [7]; doing so for a collection of families is tricky for several reasons. First-of-all merely specifying elements is ambiguous: the elements of which set and belonging to which family? Furthermore, we also need to define structural constraints; we have to specify the characteristics of elements belonging to some sets as well as the relationships between the sets in a family. In a family we cannot use information such as the position of the elements in a set or the linear order between the subsets of some sets. If we consider a homogeneous collection of families,

[8] In the following, a family of subsets $\{A_i\}_{i \in I}$ will be indicated usiing the shorthand notation $\mathscr{A}_I$.

we could define a family with an identical structure to those in the collection, label its sets, and use these labels to specify elements. But in a collection of families, data families are heterogeneous and often we do not care about the complete structure of a family but we wish only to reference some portion of the family we care about. The NESTOR Algebra needs a mechanism to:

- Identify and manipulate an element on the basis of its value.
- Identify a set on the basis of the elements it contains.
- Define the relationships of a set with the other sets in a family.

In order to address these issues we define the concept of *pattern triple* which provides a specification of the sets and elements of interest. The pattern family is fixed for a given operation, and hence provides the needed standardization over a heterogeneous collection of families. All algebraic operations manipulate sets and elements identified by means of a pattern family.

A pattern triple, that we can indicate as $\mathrm{P} = \langle \mathscr{A}_I, F_s, F_e \rangle$, constraints each sets in two ways: The formula F_s imposes structural predicates on sets requiring each set to have structural relatives (subset, superset, direct superset, etc.) satisfying other content predicates defined in the formula F_e imposed on any set. In order to understand how the pattern triple works, we propose an example presenting a pattern triple; the pattern triple we present is drawn from the NS-F presented in the Example 2.

Example 3. *Let* $\mathrm{P} = \langle \mathscr{A}_I, F_s, F_e \rangle$ *be a pattern triple where* $\mathscr{A}_I = \{A_1, A_2\}$ *is a NS-F.*

P *is a pattern triple defining a NS-F composed by two sets* A_1, A_2 *where* A_1 *is required to be the direct superset of* A_2 *and* A_1 *must not have any superset – i.e.* $\mathscr{S}^-_{\mathscr{A}}(A_1) = \{\emptyset\}$*; these conditions are expressed by means of structural predicates. On the other hand, by means of content predicates we require* A_2 *to contain an element whose label starts with the string:* `letter`*. We can see a graphical representation of the pattern family described by the pattern triple in this example in Figure 8.*

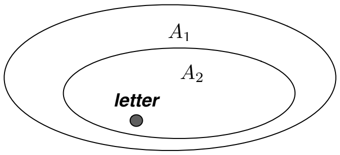


Fig. 8 A graphical representation of the pattern triple P of Example 3

Now, we can introduce another important concept of the NESTOR Algebra: the order embedding of a pattern triple P into a collection of families $\mathscr{C}$. It is useful to point out that $\mathscr{C}$ can be composed by one or more families of sets. In the following we indicate with $B_k \in \mathscr{C}$ a generic set in the collection meaning that $\exists \{B_i\}_{i \in I} \in \mathscr{C} \mid B_k \in \{B_i\}_{i \in I}$.

Let $\mathscr{C}$ be a collection of families of sets, and $\mathrm{P} = \langle \mathscr{A}_I, F_s, F_e \rangle$ be a pattern triple. An **order-embedding** [14] of P into $\mathscr{C}$ produces as output a collection of families of sets where each family has the same structure of $\mathscr{A}_I$ and satisfies the structural and content predicates defined in the pattern triple.

The families of sets outputted by the order-embedding of a pattern family into a collection of families are called *witness families*. There may be no, one or more than one embedding of a pattern triple into a collection of families and each embedding induces a *witness family* of the embedding.

A witness family is composed of each set in the input collection $\mathscr{C}$ that matches a set in the pattern triple and if there exists a set $B_t \in \mathscr{C}$ such that it is both a subset of a set that matches a pattern set and superset of another set matching a pattern set, then B_t belongs to the witness family even if it does not match any pattern set.

The meaning of "witness family" is that the sets in an instance that satisfies the pattern triple are retained and the original family structure is restricted to the retained sets to yield a witness family. If a given pattern triple can be embedded in an input family instance in multiple ways, then multiple witness families are obtained, one for each order-embedding as we show in the next example.

Example 4. *In this example we show some order-embeddings of the pattern triple presented in the Example 3 into the NS-F presented in the Example 2. This pattern triple can be embedded in three ways into the input family and then it returns three different witness families. We can see a graphical representation of the witness NS-families in Figure 9.*

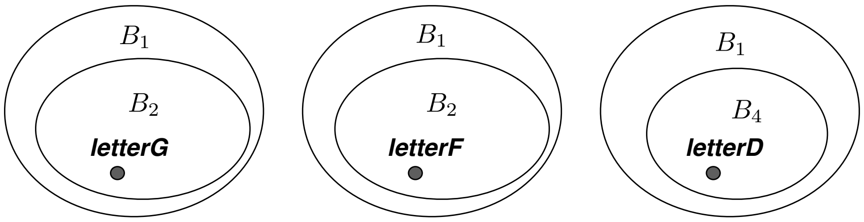


Fig. 9 A graphical representation of the witness families resulting from the pattern triple and the collection of families in Example 4

Most-of-all the operators in the NESTOR Algebra are based on the concepts of pattern triple and witness family. As an example we present the selection operator that exploits the concept of pattern triple and witness family; furthermore, we introduce the concept of *adornment list* which is a list containing pattern sets: let $\mathscr{A}_I$ be a pattern family then the adornment list is $\mathrm{SL} = \{A_j\}$ for some $A_j \in \mathscr{A}_I$. The **selection operator** takes a family of subsets (or a collection of families) as input, a pattern triple and an adornment list as parameters and it outputs a witness family for each embedding of the pattern triple in the input family; the witness families

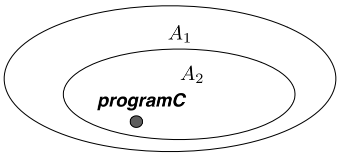


Fig. 10 A graphical representation of the pattern triple P of Example 5

produced as output are then augmented with all the subsets of each set in the adornment list even if these subsets do not match any set in the pattern triple. This means that for each set A_k in the adornment list we have to insert all the sets in $\mathscr{S}^+_{\mathscr{A}}(A_k)$ in each witness family embedded by the pattern triple. If the adornment list is empty, the selection operator straightforwardly returns the witness families. The selection operator works in the same way for the NS-M and the INS-M.

Figure 9 presents the output of the selection operator when the adornment list is empty. Let us see another example about how the selection operator can be used.

Example 5. *Let us consider the NS-F $\mathscr{B}_J$ presented in Example 2 and shown in Figure 7. Let us consider the pattern triple* $\mathrm{P} = \langle \mathscr{A}_I, F_s, F_e \rangle$ *where* $\mathscr{A}_I = \{A_1, A_2\}$ *is a NS-F. The structural predicates in F_s say that $A_1 \subseteq A_2$ and that A_1 must not have any superset. The content predicates in F_e say that A_2 must contain an element which label is "programC". This pattern is represented in Figure 10.*

If we match the presented pattern triple with the input family $\mathscr{B}_J$ we obtain a single witness family as output because there is only one configuration in the input family for which we have a set with no superset which contains a set at which belongs an element with label "programC". This witness family is presented in Figure 11.

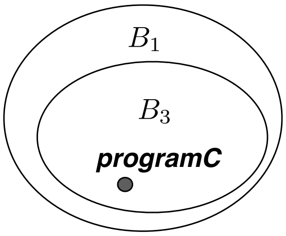


Fig. 11 A graphical representation of the witness family resulting from the pattern triple and the collection of families in Example 5

If we consider a selection operator which takes the family $\mathscr{B}_J$ as input and the just described pattern triple and an empty adornment list as parameters, the output will be exactly the witness family in Figure 11. Let us consider the same selection operator with a non-empty adornment list – i.e. $\textsc{sl} = \{A_2\}$. *This means that all the* *subsets of the set matching A_2 in the input collection will be added to the resulting witness family. So, in this example we have to augment the witness family in Figure 11 with all the subsets of set B_3; we can see the output of the selection in Figure 12.*

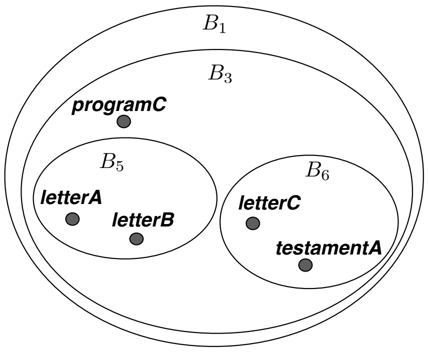


Fig. 12 A graphical representation of the output of the selection operator presented in Example 5

The other operators in the NESTOR Algebra are defined on the same theoretical basis of the selection.

6 The NESTOR Prototype: Addressing Interoperability for Digital Archives

The NESTOR Prototype is the actual instantiation of the NESTOR Model; in Figure 13 we can see the main components of the NESTOR Prototype. The first component details how the entities and the information objects we are considering are represented through the NESTOR Model; the second describes the possible instantiations of the model in an actual environment and the third examines the relationships between the instantiations and the technologies of choice.

The NESTOR Prototype presents two possible applications of the NESTOR Model. The first application shows how an archive modeled through the NESTOR Model can be instantiated by means of the EAD metadata format. We show how the archival hierarchy and the context of archival descriptions can be retained by means of an XML file. The second application shows how we can access and share archival descriptions with variable granularity by means of the OAI-PMH while retaining the fundamental characteristics of the archives. First of all we will show how an archive can be modeled through a NS-F and a INS-F; then, we analyze the requirements of interoperability in the archival context and afterwards we present the two applications of the NESTOR Model and describe how they can or cannot address the interoperability aspects for Digital Libraries.

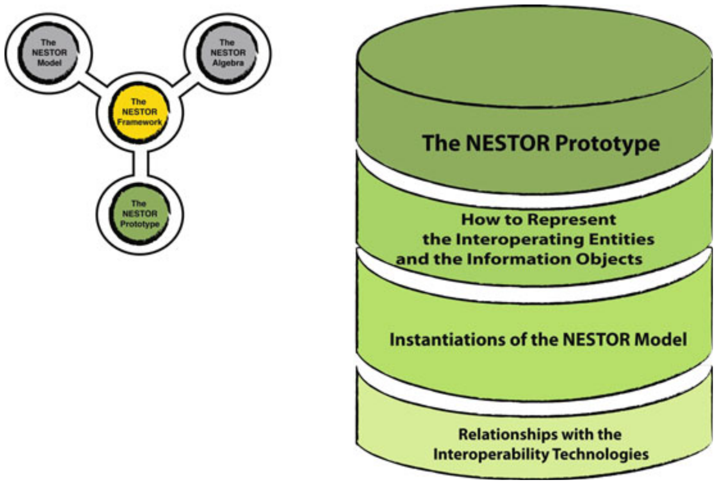

The NESTOR Model
The NESTOR Algebra
The NESTOR Framework
The NESTOR Prototype
The NESTOR Prototype
How to Represent the Interoperating Entities and the Information Objects
Instantiations of the NESTOR Model
Relationships with the Interoperability Technologies

Fig. 13 The main components of the NESTOR Prototype

6.1 How to Represent an Archive through the NESTOR Model

At this stage it is quite straightforward to model a digital archive throughout the set data models defined in the NESTOR Model. Let us consider an archive composed of several divisions – e.g. fonds, sub-fonds, series – each division contains a bunch of records – e.g. letters, registers, testaments; a representation of such an archive is given in Figure 7 where we represented an archive through a NS-F. In order to model this archive we have to represent the hierarchical relationships between its divisions and the records belonging to them. By adopting the NESTOR Model we represent each division as a set maintaining the hierarchical relationships by means of the inclusion order defined between the nested sets. Let us consider another example: if a fonds contains three sub-fonds, then in the NS-M we will have a set representing the fonds containing three subsets representing the subfonds. Vice versa in the INS-M we will have a set representing the fonds which is the common subset of the three sets representing the sub-fonds. Each record belonging to a division is represented as an element belonging to the set corresponding to that division. In this context we consider each element is a metadata – defined in whatever format – describing the archival resource; please note that the model does not necessarily require that the elements be metadata, they can also be full content digital objects represented as well as elements belonging to sets; we have seen in the NESTOR Algebra that the value of an element can be of whatever domain we want. At this level of definition there are no constraints on the nature of the elements belonging to the sets.

In Figure 2 we have seen a sample archive represented by a tree mapped into an EAD XML file; in Figure 14 below, we can see the same archive represented with the Nested Set Model and the Inverse Nested Set Model. Please note that with these models the records belonging to each archival division are properly represented as elements. In the tree there is not a defined way of representing the records; indeed, as an expedient, we represented them as a bunch of documents linked to a division by a dotted arrow. In Figure 14 we focus on the sets order inclusion and we do not indicate the label or the value of the elements just as in Figure 2 we did not specify the elements contained by the archival divisions encoded in EAD.

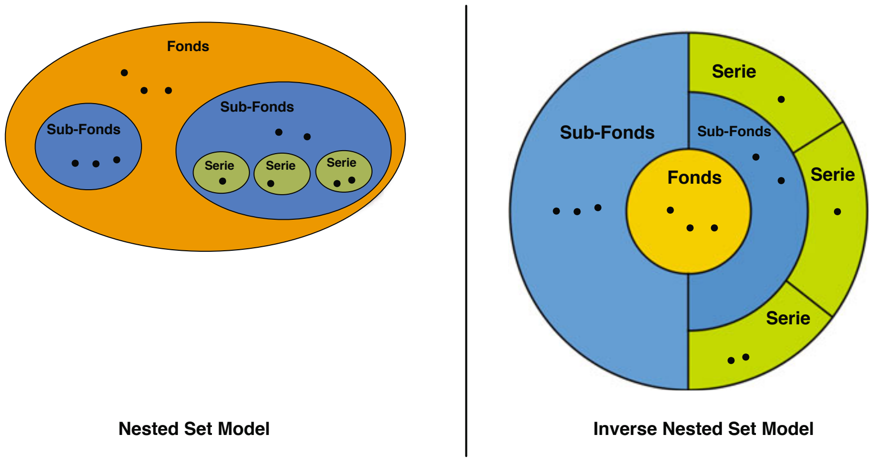


Fig. 14 A sample archive represented by means of the Nested Set Model and the Inverse Nested Set Model

6.2 *Analysis of the Requirements*

In the definition of a data model it is important to define the requirements that the instantiations of the data model have to fulfill. For the purposes of this chapter we point out the requirements that if fulfilled allow interoperability issues for the digital archives to be addressed.

> **R1.** The archival descriptions have to be accessible from multiple entry points and at the same time they have to disclose their relationships allowing the user to consult contextual information. Furthermore, the users must have a means at their disposal for manipulating both the archival structure and the archival descriptions and for defining and performing queries on-the-fly.

This requirement is important because it says that we have to be able to consult an archive starting from the required description without having to navigate the whole archival hierarchy from a unique entry point to find the information of interest. At the same time from each description we have to be able to reconstruct its relationships with the other elements of the archive – i.e. preserving and exploiting the archival bonds. When we manipulate and query an archive we have to be able to express constraints on the structure and on the content of an archive; to do so we need to have a well-defined mechanism that allows us to express our needs in a standard way. This requirement if fulfilled addresses the "*user perspective*" aspect of interoperability.

R2. The archival descriptions have to be shareable in a distributed environment with a variable granularity and have to provide a mechanism for reconstructing the hierarchy when necessary.

This requirement states that we have to be able to exchange archival descriptions with different degrees of coarseness and belonging to whatever level of the archival hierarchy without having to exchange the whole archive. Furthermore, a mechanism needs to be available for reconstructing the archival relationships of an exchanged description whenever it is necessary. In the current state of development of DL an important technological requirement for such a data model is compliance with OAI-PMH. Through the fulfillment of this requirement we can address the "*functional perspective*" and the "*interoperability technology*"interoperability technology aspects of interoperability.

R3. Advanced language techniques allowing cross-language access to the resources have to be straightforwardly applicable to the archival descriptions.

Cross-language access to information leads to problems of both semantic and syntactic interoperability [20]. Two "metadata-related challanges" need to be addressed that usually are faced by involving the specification of the language of the metadata fields" [20]: false friends and term ambiguity. Another important issue is "name resolution" which regards the necessity to disambiguate between words that are proper names that do not require a translation or nouns that have to be translated for multilingual purposes. For instance, the term "Bush" can be seen as the surname of a former president of the United States or as a noun indicating a shrub. On the other hand, we may need to translate some proper names; for instance, the proper name"Kepler" has to be translated as "Keplero" in Italian.

To address these issues we can point out three main solutions. In the **Translation** technique a query formulated into the user language is automatically translated in the other languages supported and then submitted to the system. This solution is not free from the false friends, name resolution and term ambiguity issues. The **Enrichment of Metadata** is understood as making the intended meaning of information resources explicit and machine-processable, thus allowing machines and humans to better identify and access the resources. The language would be thus provided in the metadata itself. Lastly, the **Association to a Class** is the association of terms to a fairly broad class in a library classification system such as the Dewey Decimal Classification. This is a common solution for the term ambiguity problem and it is similar to synsets used in WordNet[9]. More advanced language techniques such as semantic annotation and tagging may also be taken into account and related to this solution.

The specification of the language of metadata field permits us to fully exploit metadata for cross-language purposes. If metadata do not come with or cannot be

[9] `http://wordnet.princeton.edu/`

enriched with the language of the field, it is useful to rely on the association to a class technique. This useful technique relies on the use of the subject field of metadata; it is not always possible to determine the subject of a metadata or of a term. This is particularly true for archival metadata where determining the subject can be very difficult. This requirement is directly related to addressing the "*multilingualism*" aspect of interoperability.

6.3 Retaining Archival Hierarchy and Context throughout an XML Tree

The EAD metadata format is the standard means for representing and encoding an archive. In Figure 2 we have seen how a tree is mapped into an EAD file; in Figure 15 we can see how the same sample archive modeled by means of the Nested Set Model (see Figure 14) can be mapped in the same EAD file. The order inclusion between the sets defining the hierarchical relationships between the archival divisions is retained in the EAD by means of nested tags in the XML file. The elements representing the archival descriptions are encoded by a sub portion of XML nested inside each tag representing the corresponding archival division.

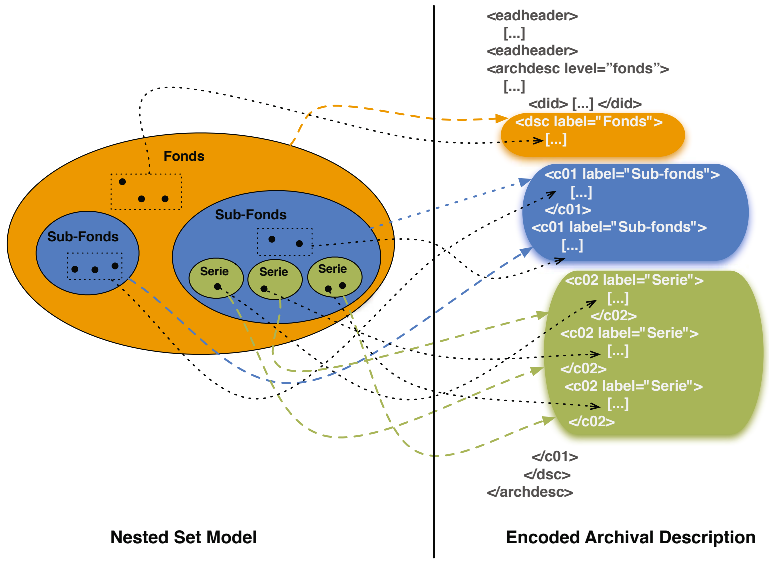


Fig. 15 A sample archive represented throughout the NSM and mapped into an EAD file

The main feature of this instantiation of the model is that both the structural and the content elements are represented by means of XML elements (i.e. tags). The EAD metadata allows us to encode the description part defined by means of the data models and thus it is a proper means for representing an archive. Let us see how it behaves with the three requirements we pointed out in the previous section.

The first requirement (**R1**, Section 6.2) states that we have to be able to access the archival description – i.e. descriptive metadata – at different degrees of coarseness. EAD is encoded as a unique XML file which mixes structural and content information while the entry point to access the information is the root of the XML file. From the root we have to navigate the hierarchy to access the information of interest. In order to overcome this issue we can define some superstructures to the EAD; for instance, we can settle some predefined entry points by the use of XPointers[10] pointing to specific elements of the XML or by using predefined paths driving the user through the hierarchical structure. These solutions are palliatives because they can only adequately match a well-defined reality with limited and specific needs; moreover, they have to be revised and adapted when the user needs or the requirements change. Furthermore, for each instantiation of the EAD, we have to know in advance how the XML elements are used; this is not a problem in general because we can make use of the EAD schema, but to do so each instantiation of EAD has to meet stringent best practice guidelines [26] otherwise the use of tags may be inconsistent, leading to wrong interpretations of the information as has happened in practice [18]. This peculiar aspect is problematic also from the manipulation and query point of view. We can adopt the Tree Algebra for XML [16] as a natural way to manipulate and query the EAD file and consider both structure and content of the encoded archive. The users can express their need throughout algebra operators, but they have to know in advance how the EAD elements are used otherwise the algebra operators are ineffective.

The second requirement (**R2**, Section 6.2) states that we have to be able to exchange archival descriptions in a distributed environment. The same issues affecting EAD for the access requirement can be found here for metadata exchange; indeed, the encoding of all the archival descriptions as a unique XML file forces us to exchange the archive as a whole. We cannot share a specific piece of information – e.g. the descriptions of the documents belonging to a specific *serie* – without extracting it from the XML file and losing in this way the structural information retained thanks to the nested tags in the XML itself.

The third requirement (**R3**, Section 6.2) regards the possibility of using language techniques with the archival metadata. The first language technique (e.g. "translation") is not affected by the choice of EAD and it can be directly applied. Furthermore, when we consider the translation of an EAD file we have the advantage of a big file with a large amount of contextual information which can be used to disambiguate the terms. On the other hand, the "enrichment of metadata" technique requires the metadata to be machine-readable in order to be automatically processed and enriched. The very flexibility of EAD leading to a not always consistent use of structure and content elements precludes the possibility of adopting this technique with many EAD files. Lastly, we know that a single EAD metadata is used to describe an entire archive, thus in a single metadata we can find very different subjects.

[10] `http://www.w3.org/TR/xptr-framework/`

With this organization it is very difficult to disambiguate terms or to identify the subject of metadata; with the EAD metadata the "association to a class" technique is essentially unworkable.

We can see that in this case we are not able to meet the interoperability requirements for a digital archive. We can think of different solutions that address one specific aspect at a time. These solutions must be designed on a case-by-case basis and they do not constitute a general environment that can be applied to hierarchically structured resources for addressing interoperability in Digital Libraries.

6.4 Encoding, Accessing and Sharing an Archive through Sets

This application of the NESTOR Framework follows a different approach and it is based on the joint use of some basic features of OAI-PMH and the Dublin Core metadata.

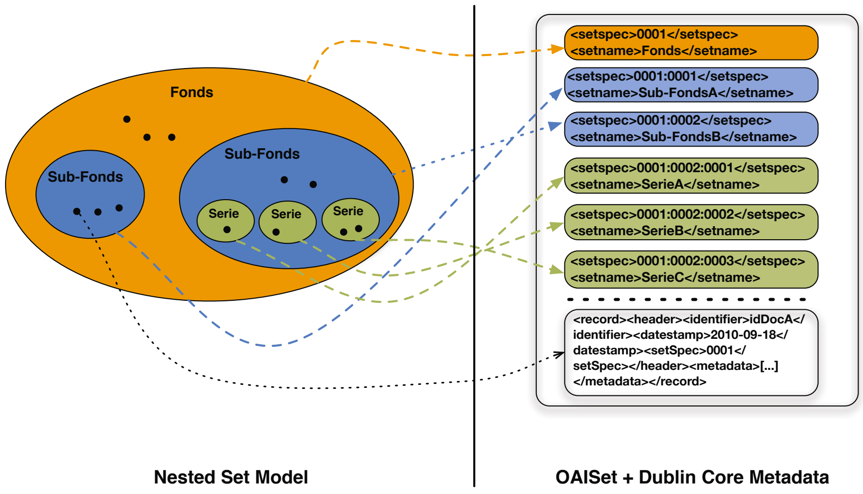

Fonds
Sub-Fonds
Sub-Fonds
Serie
Serie
Serie
<setspec>0001</setspec> <setname>Fonds</setname>
<setspec>0001:0001</setspec> <setname>Sub-FondsA</setname>
<setspec>0001:0002</setspec> <setname>Sub-FondsB</setname>
<setspec>0001:0002:0001</setspec> <setname>SerieA</setname>
<setspec>0001:0002:0002</setspec> <setname>SerieB</setname>
<setspec>0001:0002:0003</setspec> <setname>SerieC</setname>
<record><header><identifier>idDocA</identifier><datestamp>2010-09-18</datestamp><setSpec>0001</setSpec></header><metadata>[...]</metadata></record>
Nested Set Model
OAISet + Dublin Core Metadata

Fig. 16 A sample archive represented throughout the NSM and mapped into OAI sets and DC metadata

Now we informally discuss how the set data models are mapped into a structure of OAI sets and Dublin Core metadata; a formal definition of the ideas behind this approach can be found in [1, 11]. First-of-all we present the mapping of an archive represented by means of a NS-F into an organization of OAI sets and DC metadata; the mapping of the INS-F is symmetrical to this procedure but it leads to a slightly different outcome [1]. Let us consider the sample archive represented by the Euler-Venn diagram in Figure 14. As we can see in Figure 16, each set composing this nested set structure is mapped into an OAI set with a proper `setSpec`; the set called "fonds" is mapped into an OAI set with $<$ `setspec` $>$ `0001` $<$ /`setspec` $>$. This set has two subsets that are mapped into two OAI sets: $<$ `setspec` $>$ 0001 : 0002 $<$ `/setspec` $>$ and $<$ `setspec` $>$ 0001 : 0003 $<$ `/setspec` $>$ and so on for the other sets. We can see that the hierarchical relationships and thus the inclusion order between the sets is maintained by the identifiers of the OAI sets which are defined as materialized paths from the root to the identified set. Each single archival description is mapped into a DC metadata belonging to an OAI set; the membership information is added to the header of these metadata that are seen as OAI-records. In this way each archival description can be encoded by a single metadata without any constraints on its format; indeed, an OAI set can contain different kinds of metadata formats. With this model we do not impose any conditions on the archival descriptions, thus allowing the possibility of changing the metadata, updating the information or adding a new metadata format without affecting the structure of the archive and without changing the data model. The choice of the DC metadata format is lead by its widespread use in libraries and the possibility of defining Dublin Core *application profiles* which allow us to make it domain-specific; indeed, DC application profiles allow the definition of DC metadata formats well-suited for the reality we intend to represent.

This instantiation of the set data models has two main differences from the EAD one: it clearly divides the structural elements (i.e. the sets) from the content elements (i.e. the archival descriptions) and it does not bind the archival descriptions to a unique, fixed and predefined metadata format. These differences have a major impact on the fulfillment of the three presented requirements.

The **R1** requirement is fulfilled because each OAI set is individually accessible as well as each single metadata. From a set we can easily reconstruct the relationships with the other sets by exploiting the `setspec`; from a metadata we can reconstruct the relationships with the other metadata thanks to the membership information contained in their header. At the same time we can straightforwardly adopt the NESTOR Algebra to manipulate and query the archival descriptions. Indeed, each set is uniquely identified by a `setspec` value and the name of the set is a mandatory requirement; the metadata are encoded by means of the Dublin Core and thus the use of the tags is simple and consistent. By means of the pattern triple we can express requirements on the structure of the archive (i.e. the nested sets by means of the pattern family and the boolean formula F_s) and on the archival descriptions (i.e. the metadata by means of the boolean formula F_e). The NESTOR Algebra gives users a standard way to manipulate and query hierarchical structure that can be applied to different *interoperating entities*; users do not have to change the manipulation and query language and thus they can perform a query both in the library and the archival context in the same way. The NESTOR Algebra defines a natural way to modify the structure and the content of an archive represented with the NESTOR Model; at the same time we can query an archive naturally in the context of the NESTOR Framework.

Let us consider the **R2** requirement; throughout the OAI sets and DC metadata approach we can easily use OAI-PMH to exchange a single set or a single metadata, thus allowing a variable granularity exchange. Furthermore, from the identifier of an OAI set we can reconstruct the hierarchy through the ancestors to the root. By means of OAI-PMH it is possible to exchange a specific part of the archive while at the same time maintaining the relationships with the other parts of it. The NS-M fosters the reconstruction of the lower levels of a hierarchy; thus, with the couple NS-M and OAI-PMH applied to the archive, if a harvester asks for an OAI set representing for instance a sub-fonds it recursively obtains all the OAI subsets and items in the subtree rooted in the selected sub-fonds.

It is worthwhile highlighting that this approach can also be applied with the INS-M, then if a harvester asks for an OAI set representing for instance an archival serie, it recursively obtains all the OAI-subsets and records in the path from the archival serie to the principal fonds that is the root of the archival hierarchy. The choice between a NS-M or INS-M should be made on the basis of the application context. In the archival context the application of the INS-M would be more significant than the NS-M. Indeed, often the information required by a user is stored in the external nodes of the archival tree [28]. If we represent the archival tree by means of the INS-M, when a harvester requires an external node of the tree it will receive all the archival information contained in the nodes up to the root of the archive. This means that a Service Provider can offer a potential user the required information stored in the external node and also all the information stored in its ancestor nodes. This information is very useful for inferring the context of an archival metadata which is contained in the required external node; indeed, the ancestor nodes represent and contain the information related to the series, sub-fonds and fonds in which the archival metadata are classified. The INS-M fosters the reconstruction of the upper levels of a hierarchy which in the archival case often contain contextual information which permit the relationships of the archival documents to be inferred with the other documents in the archive and with the production and preservation environment. We can see how the possibility of changing from one set data model to the other by means of the defined mapping functions is very useful in the archival context; we can address the user requirements in the most effective way without being bound to the properties of the model of choice.

With regard to the **R3** requirement, we can see that this approach is particularly well-suited for use in conjunction with the presented language techniques. Indeed, the representation of an archive as an organization of sets and DC metadata makes it easier to determine the subject of each single metadata and thus to apply the "association to a class" solution; in the same way the metadata enrichment can be adopted because the DC metadata are well-suited to automatic processing. In this way the solutions proposed to enable cross-language access to digital contents can be applied also with the archival metadata, thus opening up these valuable resources to a significant service offered by the Digital Library technology.

7 Conclusions and Future Work

The NESTOR Framework has been introduced as a conceptual and logical environment that can be exploited to address interoperability issues in Digital Libraries. The NESTOR Framework focuses on hierarchical structured resources by proposing two set data models alternative to the tree data structure which, as we have seen in some application contexts are well-suited to addressing interoperability issues. Furthermore, we presented the NESTOR Algebra that allows us to manipulate and query the hierarchies represented through the NESTOR Model in a natural way. We presented a concrete use case based on archives, which are a fundamental and challenging entity in the Digital Libraries panorama. Within the archives we showed how an archive can be represented through set data models and how these models can be instantiated. We compared two instantiations of the NESTOR Model dealing with issues of interoperability for Digital Libraries. We showed that the use of sets to express archives open them up to new important functions that meet interoperability issues.

An archival information management system has been envisioned and it is under design and development to implement the NESTOR Model for the modeling and managing of archival resources. In the design of the system, it has been possible to separate the structural representation from the content representation of the archival resources thanks to the features of the NESTOR Model. The design of a system that allows for the separation of the two levels of representation makes a step forward in the area where the available management systems do not allow such a level of abstraction in the possible application implementations.

The archival information management system, under development, is going to be adopted in the next future in the context of a project of the Italian Veneto Region[11]. Main aim of the project is to make available a regional archival information system, which allows the management of the resources of archives that are present on the region territory.

Acknowledgments. The work reported has been envisaged in the context of an agreement between the Italian Veneto Region and the University of Padua that aims at drawing up innovative information management solutions for improving the final user access to archives.

EuropeanaConnect[12] (Contract ECP-2008-DILI-52800) and the PROMISE network of excellence[13] (contract n. 258191) projects, as part of the 7th Framework Program of the European Commission, have partially supported the reported work.

References

1. Agosti, M., Ferro, N., Silvello, G.: Access and Exchange of Hierarchically Structured Resources on the Web with the NESTOR Framework. In: IEEE/WIC/ACM International Conference Web Intelligence and Intelligent Agent Technology, pp. 659–662 (2009)
2. Agosti, M., Ferro, N., Silvello, G.: The NESTOR Framework: Manage, Access and Exchange Hierarchical data Structures. In: Proceedings of the 18th Italian Symposium on Advanced Database Systems, Società Editrice Esculapio, Bologna, Italy, pp. 242–253 (2010)

[11] `http://www.regione.veneto.it/`
[12] `http://www.europeanaconnect.eu/`
[13] `http://www.promise-noe.eu/`

3. Anderson, K.W., Hall, D.W.: Sets, Sequences, and Mappings: The Basic Concepts of Analysis. John Wiley & Sons, Inc., New York (1963)
4. Bender, M.A., Farach-Colton, M., Pemmasani, G., Skiena, S., Sumazin, P.: Lowest Common Ancestors in Trees and Directed Acyclic Graphs. J. Algorithms 57, 75–94 (2005)
5. Celko, J.: Joe Celko's SQL for Smarties: Advanced SQL Programming. Morgan Kaufmann, San Francisco (2000)
6. Codd, E.F.: A Relational Data Model of Data for Large Shared Data Banks. Communications of the ACM 13(6), 377–387 (1970)
7. Codd, E.F.: Relational Completeness of Data Base Sublanguages. In: Rustin, R. (ed.) Database Systems, pp. 65–98 (1972)
8. Cook, T.: The Concept of Archival Fonds and the Post-Custodial Era: Theory, Problems and Solutions. Archiviaria 35, 24–37 (1993)
9. Crestani, F., Vegas, J., de la Fuente, P.: A Graphical User Interface for the Retrieval of Hierarchically Structured Documents. Inf. Process. Management 40(2), 269–289 (2004)
10. Duranti, L.: Diplomatics: New Uses for an Old Science. Society of American Archivists and Association of Canadian Archivists in association with Scarecrow Press (1998)
11. Ferro, N., Silvello, G.: The NESTOR framework: How to handle hierarchical data structures. In: Agosti, M., Borbinha, J., Kapidakis, S., Papatheodorou, C., Tsakonas, G. (eds.) ECDL 2009. LNCS, vol. 5714, pp. 215–226. Springer, Heidelberg (2009)
12. Gilliland-Swetland, A.J.: Enduring Paradigm, New Opportunities: The Value of the Archival Perspective in the Digital Environment. Council on Library and Information Resources (2000)
13. Gradmann, S.: Interoperability of Digital Libraries: Report on the work of the EC working group on DL interoperability. In: Seminar on Disclosure and Preservation: Fostering European Culture in The Digital Landscape, National Library of Portugal, Directorate-General of the Portuguese Archives, Lisbon, Portugal (2007)
14. Halmos, P.R.: Naive Set Theory. D. Van Nostrand Company, Inc., New York (1960)
15. International Council on Archives. ISAD(G): General International Standard Archival Description, 2nd edn. International Council on Archives, Ottawa (1999)
16. Jagadish, H.V., Lakshmanan, L.V.S., Srivastava, D., Thompson, K.: TAX: A tree algebra for XML. In: Ghelli, G., Grahne, G. (eds.) DBPL 2001. LNCS, vol. 2397, pp. 149–164. Springer, Heidelberg (2002)
17. Kiesling, K.: Metadata, Metadata, Everywhere - But Where Is the Hook? OCLC Systems & Services 17(2), 84–88 (2001)
18. Kim, J.: EAD Encoding and Display: A Content Analysis. Journal of Archival Organization 2(3), 41–55 (2004)
19. Knuth, D.E.: The Art of Computer Programming, 3rd edn., vol. 1. Addison-Wesley, Reading (1997)
20. Levergood, B., Farrenkopf, S., Frasnelli, E.: The Specification of the Language of the Field and Interoperability: Cross-Language Access to Catalogues and Online Libraries (CACAO). In: Greenberg, J., Klasore, W. (eds.) DC 2008, Proc. of the Int'l Conf. on Dublin Core and Metadata Applications 2008, pp. 191–196. Universitätsverlag Göttingen, Germany (2008)

21. MacNeil, H., Wei, C., Duranti, L., Gilliland-Swetland, A., Guercio, M., Hackett, Y., Hamidzadeh, B., Iacovino, L., Lee, B., McKemmish, S., Roeder, J., Ross, S., Wan, W., Zhon Xiu, Z.: Authenticity Task Force Report. InterPARES Project: Vancouver, Canada (2001)
22. Paparizos, S., Jagadish, H.V.: The importance of algebra for XML query processing. In: Grust, T., Höpfner, H., Illarramendi, A., Jablonski, S., Fischer, F., Müller, S., Patranjan, P.-L., Sattler, K.-U., Spiliopoulou, M., Wijsen, J. (eds.) EDBT 2006. LNCS, vol. 4254, pp. 126–135. Springer, Heidelberg (2006)
23. Pearce-Moses, R.: Glossary of Archival And Records Terminology. Society of American Archivists (2005)
24. Pitti, D.V.: Encoded Archival Description. An Introduction and Overview. D-Lib Magazine 5(11) (1999)
25. Prom, C.J.: Does EAD Play Well with Other Metadata Standards? Searching and Retrieving EAD Using the OAI Protocols. Journal of Archival Organization 1(3), 51–72 (2002)
26. Prom, C.J., Rishel, C.A., Schwartz, S.W., Fox, K.J.: A Unified Platform for Archival Description and Access. In: Rasmussen, E.M., Larson, R.R., Toms, E., Sugimoto, S. (eds.) Proc. 7th ACM/IEEE Joint Conference on Digital Libraries (JCDL 2007), pp. 157–166. ACM Press, New York (2007)
27. Rys, M., Chamberlin, D.D., Florescu, D.: XML and Relational Database Management Systems: The Inside Story. In: Özcan, F. (ed.) Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2005), pp. 945–947. ACM Press, New York (2005)
28. Shreeves, S.L., Kaczmarek, J.S., Cole, T.W.: Harvesting Cultural Heritage Metadata Using the OAI Protocol. Library Hi Tech 21(2), 159–169 (2003)
29. Van de Sompel, H., Lagoze, C., Nelson, M., Warner, S.: Implementation Guidelines for the Open Archive Initiative Protocol for Metadata Harvesting - Guidelines for Harvester Implementers. Technical report, Open Archive Initiative, p. 6 (2002)
30. Van de Sompel, H., Lagoze, C., Nelson, M., Warner, S.: The Open Archives Initiative Protocol for Metadata Harvesting, 2nd edn., Technical report, Open Archive Initiative, p. 24 (2003)
31. Vegas, J., Crestani, F., de la Fuente, P.: Context Representation for Web Search Results. Journal of Information Science 33(1), 77–94 (2007)