ICCD2OWL: an ICCD to OWL data compiler

[1]A. Aiello, [1]S. Brandi, [1]M. Mango Furnari and [2]F. Proto

[1]Istituto di Cibernetica "E. Caianiello"- CNR
E-mail: {a.aiello, mf }@cib.na.cnr.it s.brandi@remuna.org
[2]Dipartimento di Discipline Storiche "E. Lepore" Università Federico II di Napoli
E-mail: fiona.proto@virgilio.it

Abstract— **This paper outlines the problems the authors came across when, after the realization of ReMuNaICCD, an ontology supposed to allow a more articulate use of the present cultural heritage information system, they approached the task of populating it with instances.**

To permit immediate use of the ontology, it was necessary to develop ICCD2OWL, a data compiler able to translate the data recorded in the available ICCD cards into class instances and property instances for the ontology. The first task was to design formal codifications for both the ICCD attribute-value data model and the ReMuNaICCD ontological data model. Then a program for converting each of them into the other was required.

Actually, the issue faced in this paper comes under the wider topic of the mappings between simple attribute-value data structures and ontological representations of the knowledge or, in more general terms, between not-object oriented models and object oriented models.

The most remarkable conclusion is that part of the effort required is due to the semantic inadequacies of the ontology languages currently available. Such inadequacies make any attempt at accomplishing semantic interoperability based on those languages alone unproductive. Therefore, it is evident that the methodologies and results presented here could profitably be used in a more general framework like the Semantic Web.

I. INTRODUCTION

In designing ReMuNaICCD, a formal ontology for the Cultural Heritage domain [1], we had as fundamental reference the recommendations issued by the Central Institute for the Catalogue and Documentation (ICCD). The research was supported by the Virtual Museum of Naples project ReMuNa[1] (which stands for Network of Neapolitan Museums) and SIABeC[2] (Information System Applied to the Cultural Heritage).

This project was motivated by the necessity of having an ontology model compatible with the information system currently deployed inside the institutions that are in charge of the safekeeping, maintenance and valorisation of the cultural assets.

To this end, the ICCD cards that describe the material assets and those that describe the immaterial assets, and not least those related to the authority files and to the multimedia documentation, have been investigated. Moreover, considering their importance in achieving a comprehensive description of the excavations, recognitions and archaeological essays, the cards that were structured by the ICCD in order to document the stratigraphic units have also been studied and transferred into the ontology.

It was necessary to make a careful study of the ICCD cards. First of all, in order to distinguish two macro-groups of fields: those giving information about the object the card is devoted to, and those giving information on the card itself. Inside these two macro-groups, we have characterized the fields designed to supply identifications and characterizations, the fields designed to report events, and the fields that determine relations with other elements (documents, other assets, other cards etc).

In order to establish which classes had to be created, we have distinguished on one hand, fields that appear without variations in all ICCD cards, from those designed for taking into account of specific typologies of assets. On the other hand, we have determined, for every field, whether it was directly or indirectly related to the subject catalogued by the card.

ReMuNaICCD is not in any sense a simple ICCD fields transposition into an object oriented model. In order to represent all the aspects of the cultural assets and the events that happened to them, it has proved to be a new data model, built from scratch.

To summarise, starting from the ICCD cards, which include 27 fields of the Bibliography card, to the nearly 300 fields of the Architectural card, with an average of 200 fields per card, we elaborated ReMuNaICCD.v2.0, an ontology of 381 classes (199 are Appellation subclasses), 473 objectProperties, 458 dataTypeProperties and about 750 instances (nearly all of them were taken from the ICCD vocabulary).

ReMuNaICCD has been written using the subset of OWL called OWL Lite [2]. OWL is the acronym of Web Ontology Language [3], a standardized language for the specification of formal ontologies, recommended by the W3C. The main factor in choosing OWL Lite was the ascertainment that not only it offers a sufficient expressivity, but also guarantees a priori computational tractability of the final product [4].

The works [5] [6] and [7] of Guarino et al. on the formal ontologies foundations, those of Gangemi et al. on the ontology patterns [8], the guide lines proposed by Alan Rector

[1] The ReMuNa project is financed by the MIUR with the Law n.488 initiative of Cluster

[2] The project SIABeC is financed with the projectCentro regionale di Competenza per lo sviluppo ed il trasferimento della innovazione tecnologica (INNOVA) P.O.R. Campania misura 3.16.

et al. [9] and the newest experiences reported online by the OEP [10], have strongly influenced the development of our ontology.

Since the start of our activities in this field, we have used Protégé as the only tool for editing our ontologies. Protégé is a free, open source ontology editor and knowledge base framework [11] [12]. Furthermore, we directly linked Protégé and the system Sesame [13] [14] for the exchange of ontologies through the Internet, by means of a plug-in [15] produced inside the project ReMuNa [16].

In section II, we introduce the main features of the ontological model: the uppermost components of the class hierarchy and the fundamental ontology pattern. Section III gives the basis for the development of the compiler. In the first subsection, the input and the output data models are compared; in the second subsection, it is explained how to use the Historicism Pattern in order to get "Conceptual Translations". Section IV, in five subsections, describes the formalisms adopted in order to a) serialize the ontology patterns, b) identify the instances, c) codify the classes, d) use global variables, and e) define a configuration file. Section V outlines the processes that the compiler performs when translating ICCD to ReMuNaICCD and viceversa. In Section VI, some conclusions are summarized and a question is asked: aren't the current ontologies' formalizations absolutely inadequate for solving the problem of semantic interoperability?

II. The Ontological Model

The ontology ReMuNaICCD has been built taking account of the most common top-level ontologies, in particular DOLCE [17] and ICOM/CIDOC-CRM [18]. Here, we introduce only a small part of the ReMuNaICCD classes and properties, just those that are involved in the topic dealt with in this paper.

The root of the classes' hierarchy is the class **Entity**. It is the class that represents the universe we are interested in and is articulated in two radically different subclasses (Fig. 1):

Concrete: the more general class that comprises all the concepts that model the domain that we have to analyze and to formally represent.

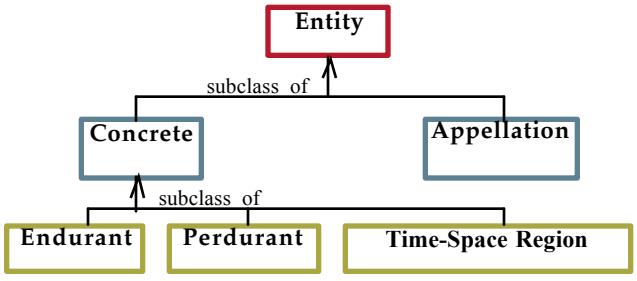

Fig. 1. The ReMuNaICCD's top-level ontology

Appellation: the root of the dictionaries that were created and are controlled by third parties, whose semantics remains alien to ReMuNaICCD.Concrete is divided into three disjoint subclasses:

The **Endurant**, i.e. the class of the subjects/objects represented by abstracting them from all possible contingent considerations, namely, attributes and relations that are understandable if and only if they make reference to certain specific space-time contexts.

The **Perdurant**, the class of the observations in which the subjects/objects emerge in a given space-time context, and exhibit a set of attributes and relations that are specific to that context.

The **Space-Time Region**, the class that models the space and time where the observations are detected.

Besides the subsumption relation, that determines the classes' hierarchy, in an ontology, classes are interlinked through a network of specific relations that formalizes not only the individual meaning of each class but also, and more interestingly, the meaning of their staying together.

It has been discovered by Gangemi et al. [8], that some not trivial "conceptual structures" of the domain prove to be represented by "semantically autonomous" clusters inside the ontological network, each involving many classes and many properties, and sometimes even meta-properties (properties of properties). Those "conceptual structures" are totally lost when the ontology is coded using the currently available languages for ontology, namely RDF/RDFS and OWL. Indeed, those languages do not offer any syntactical constructs that permit the representation of ontological substructures, and even less the essential accompanying instructions for their use. Currently, ontologies are represented as oriented graphs in which the nodes represent the classes and the arcs represent the properties. Therefore, any "conceptual structure" relevant to the domain will be represented by some pattern of connected node-arc-node triples abstracted from the whole graph. Actually, ontology patterns, made of classes and correlating properties, constitute an optimal instrument for highlighting common features and for generalizing complex relationships or decomposing them in their constitutive elements.

The Historicism Pattern illustrated in Fig. 2 gives the most important conceptual structure in the ontology ReMuNaICCD. It shows how the various kinds of historical reports are represented in terms of the fundamental classes, i.e. **Endurant**, **Perdurant**, and **Space-Time Region** and some sub properties of **comprises**, the most general objectProperty that formalizes the specific tools for performing the analysis of **Concrete**.

In this ontology pattern, **Participation**, a subclass of **Perdurant**, plays the role of the basic atomic module by means of which all the classes representing the various kinds of historical reports can be built. The constructors are some (non-transitive) sub properties of **comprises**. **Participation** formalizes the observation of a single object (**has_present = Endurant**) while it plays a certain role (**has_role = Role**), in participating in an elementary interaction (**participates_in = History_Fragment**), at a certain place and time (**has_space-time_location = Space-Time_Region)**.

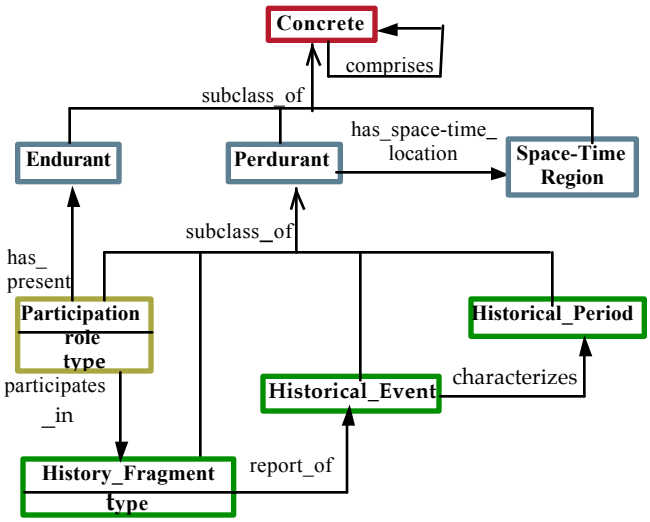

Fig. 2. The Historicism Pattern

Participation has a particular role because, through the primitive relation **has_present**, it embeds the Endurant into the space-time context (*"***Endurant present_in Perdurant***"*). Indeed, **Participation** is the class that binds the single Endurant to the attribute and the relation (i.e. the properties **has_role** and **participates_in**) that describe its manifestation in the specific space-time context indicated by **has_space-time_location**.

The class **History_Fragment**, a subclass of **Perdurant**, models the interactions. Namely, two or more instances of **Participation** are connected by the property **participates_in** to the same instance of **History_Fragment**: the **Endurant**s that are **present_in** those **Participation**s are intended to interact with each other playing the roles indicated inside their respective **Participation**s.

The class **Historical_Event** models observations which are more complex than the single **History_Fragment**; many **History_Fragment**s can be composed in a single **Historical_Event** through the property **is_report_of** and its inverse **has_report**.

Finally, every entity comprising whatever number of **Historical_Event**s is defined as **Historical_Period**.

The Historicism Pattern, through the property **role** of the class **Participation** and the property **type**, defined for any **Concrete**, can describe the conceptual structure of any kind of historical report, even reports which are much more complex than those codified by the ICCD schema.

III. From ICCD to ReMuNaICCD

A. Comparing the two data models

As already mentioned, the ICCD format[3] consists of a sequence of attribute-value pairs concerning the Cultural Heritage. The admissible values of the attributes depend on the cultural asset in question. In any case, the assignment of values to attribute is not always compulsory, just as repeated assignments of single independent values or groups of values can be permitted.

Actually, the technical specifications of the ICCD, i.e. the sequential file format, and its intrinsic characteristics, i.e. the arbitrariness of the number of the fields and the presence of functional dependences among the fields, mean that the results already reported in the literature, like many concerning relational databases, are irrelevant, making it necessary to start again from scratch.

The ontological structure described in the previous paragraph constitutes the basic skeleton of ReMuNaICCD. That structure was enriched with classes, properties and instances deduced by the various segments (paragraphs, fields and subfields) of which the ICCD cards are made up.

It must be remarked that, the modalities for determining the relations in the ICCD schema are radically different from those used in the ontological model. An example of this can be given by comparing the modalities that the two models adopt in order to deal with the relationship between the subjects/objects and the events to which they participate.

Let us consider a subject "X" that carries out one of the following two functions:

(a), the function of a scientific director in a survey which enabled the finding of a good "α"

(b), the function of a collaborator responsible for compiling an ICCD card of the good "α"

then, according to the ICCD data model:

in the case (a), the name of the subject "X" has to be reported in the simple fields of the ICCD card of the good "α", which refer to the Scientific Director of the Survey, which have enabled the finding of the good "α";

in the case (b), the name of "X" has to be reported in the fields that refer to the Collaborator responsible for the compilation of the ICCD card.

In ReMuNaICCD, instead,

in the case (a), the subject "X" is an instance of **Person**, **present_in**, an instance of **Participation**, with the attribute **role** set to **Scientific_Director**; that instance of **Participation participates_in** an instance of **History_Fragment** that is **report_of** an instance of **Survey** (subclass of **Historical_Event**);

in the case b), the subject "X" is an instance of **Person** that, **present_in** a **Participation** with the **role** of **Collaborator**, **participates_in** an instance of **History_Fragment** of the type **Compilation_of_ICCD_card**

B. The Conceptual Translations

After the foregoing analysis, which showed the basic conceptual elements of the ICCD recommendations, a complex re-composition process becomes necessary. Firstly,

[3] The technical specifications can be found at http://www.iccd.beniculturali.it/standard/

we must find the correspondences between every ICCD field and the elements that constitute our ontology. Then, every segment of the ICCD cards must be associated with some ontology pattern that connects "semantically" its ontological representative to the class describing the card's main subject.

Therefore, we need to design some formal languages (XML documents) that enable us to express: a) the correspondences 'ICCD field – ontology element', b) the ontology patterns, and c) how the compiler must use the previous data so that the mechanical translation of an ICCD card into classes, properties and instances, does in fact lead to the expected codifications.

To give an example of this process, let us consider the card for the archaeological find (RA) and its structured field reserved to the restoration data. An archaeological find, reported in the card RA is dealt with by ReMuNaICCD as an instance of the class **Archaeological_Find** (subclass of **Endurant**), while a report about a restoration is dealt with as an instance of **Restoration**, a subclass of **History_Fragment**. The ontology pattern instance shown in Fig.3 describes how **Archaeological_Find** instances are related to the ontological correspondents of the ICCD fields devoted to restorations: RSTD (date), RSTS (situation), RSTE (responsible agency), RSTN (operator) and RSTR (financing agency).

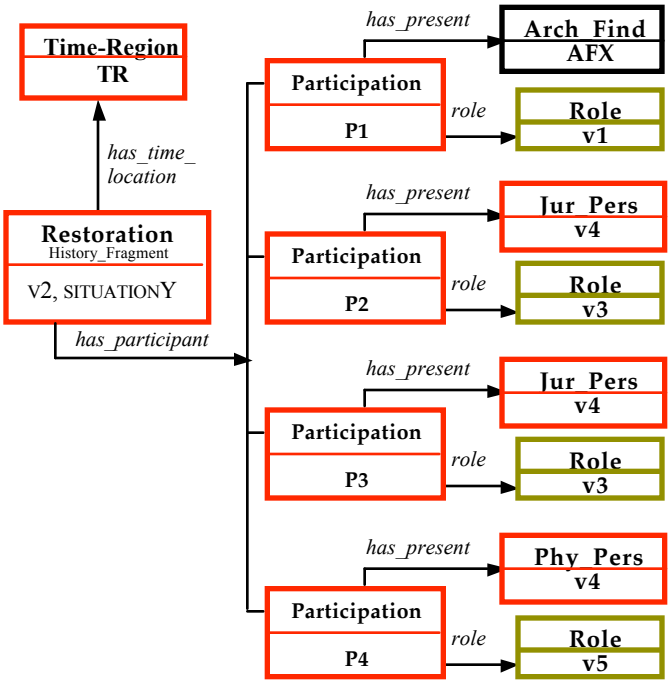

Fig. 3. Example of Historicism Pattern instance

Of course, in the graph, the boxes represent instances: the upper part of each box contains the name of the instance's class and the lower part contains the name of the instance (in the case of **Restoration**, the value **SituationY** of the property **situation** is also reported).

To transfer the contents of the fields RSTD, RSTS, RSTE, RSTN and RSTR, the data compiler has to build the instances represented by the red boxes and, at the same time, implement the mapping given in Table 1: the contents of the fields in the first column become the values indicated in the second column (references are made using the same symbols as Fig. 3); the third column gives the classes of the instances related to the values referenced in the second column. The other boxes in the pattern represent either built-in instances (in yellow) or instances (in black) that transpire from the translation of other ICCD fields.

ICCD field	Instance name or property value	**Instance Class or property**
RSTD	TR	**Time_Region**
RSTS	*situation*(of v2)	**Situation (of Restoration)**
RSTE	v4	**Juridical Person**
RSTN	v5	**Physical Person**

Table 1. Mapping ICCD fields vs ontological elements

IV. An outline of the ICCD2OWL formalism.

A. The pattern's serialization

To make a codification which is machine-understandable, we serialized all the patterns and all the rules concerning the matchings into XML documents.

In order to optimize the reuse, patterns and fields are defined separately.

Binding ICCD field/ Pattern	<iccd> <bind> <field>RSTE</field> <pattern> Arch_Find-Jur_Pers-name </pattern></bind>
Binding Parameter/ value	<bind> <param>v1</param> <value>"Restored_object"</value> </bind> <bind> <param>v2</param> <value>"Restoration"</value> </bind> <bind> <param>v3</param> <value>"Financial_Support"</value> </bind> <bind> <param>v4</param> <value>@@RSTE@@</value> </bind>
Start Class Condition: TSK="RA"	<class>Archaeological_Find</class> <condition> <contains> <field>TSK</field> <value>"RA"</value> </contains> </condition> </iccd>

Table 2. Example of ICCD field/Pattern binding

The patterns are referable by unique assigned identifiers so that each ICCD field can be bound to one or more precise

patterns. Moreover, since the same ICCD field can be bound to different patterns, depending on the context, our formalism also considers even the possibility of declaring parameters as well as conditions on the choice of the bindings.

For example, inside the ICCD, there is the field Card Type (TSK) that specifies the subject of the card, namely the "starting class". It can be an Archaeological Find or an Archaeological Site or whatever other cultural good or asset. It is clear, that on the basis of this piece of information, the choice of the bindings proves to be restricted and the parameters that must be passed are determined. The code that formalizes this choice, for the previous example, is sketched in Table 2.

The patterns are serialized as lists of triples whose elements contain parameters that assume values according to the fields of the ICCD cards. The lists may contain names of other patterns that can be considered as sub patterns of those codified by the lists. For example, the codification of the pattern Arch_Find-Jur_Pers-name quoted in Table 2 is sketched in Table 3.

Pattern Id In param. subpattern Out param. evaluating variables	<pattern> <id>Arch_Find-Jur_Pers</id> <input>v1,v2,v3,v4</input> <branch> <pattern>Arch_Find-@@v2@@ </pattern> <receive>v1,v2</receive> </branc> <branch> <pattern>Juridical_Person-@@v2@@ </pattern> <receive>v3,v4</receive> </branc> <branch> <triple> <instance_of>Juridical_Person </instance_of> <property>name</property> <value>@@v4@@</value> </triple> </branch> </pattern>

Table 3. Example of pattern serialization

B. The identity of the instances

Every time one proceeds to an object oriented representation, it is necessary to choose a means of recognizing and distinguishing the various instances of a certain class.

The simplest possible approach is that of assigning unique progressive identifiers to the instances as they are created. But this requires introducing some rules for detecting, identifiers apart, if two or more instances are equivalent, i.e. they refer to the same entity. For example, before introducing whatever new instance, it would be necessary to verify whether no instance representing the same entity is already recorded.

In designing ICCD2OWL, a different policy was preferred. Each instance whose scope is wider then one single card is endowed with an identifier both unique and significant for the entity that it represents. Every time two or more cards refer to the same entity, it is ossible to create several instances all having the same identifier: subsequently, the system is able to detect the duplications and to delete them.

The foregoing policy is implemented by coding, in the configuration document of ICCD2OWL, the properties that mostly carry identity criteria and the rules according to which starting from their values, the instances' identifiers must be composed.

C. Class use case specifications

As we have already seen, the same class can be instantiated for representing entities that receive the values of the same property from different ICCD fields. For example, an instance of the class **Physical_Person** receives the value of **name** from the field RCGA (scientific advisory) if the ICCD card reports that entity as a **Scientific Director**. Differently, in the case the entity is reported in the ICCD card as an **Authority**, then the instance of the class **Physical_Person** must receive the value of **name** from the field FUR (the collaborator responsible).

Class name inside the config file Pattern to the instance's ID (referred by v1) Class name inside ReMuNaICCD	<class> <id>Physical_Person</id> <iccd_pattern> @@PREFIX@@Phy_Pers_@@v1@@ </iccd_pattern> <name> @@PREFIX@@Physical_Person </name> </class>

Table 4. Example of the use case specification of a class

For the correct interpretation of the ICCD cards, ICCD2OWL receives the information it needs in cases like the example above by means of pieces of code like the one shown in Table 4.

D. The global variables

The fact that each ontological pattern involves more than one ICCD field, implies serious difficulties when it comes to translating one by one the fields of the same card. ICCD2OWL has to create a single pattern of instances that translate all of the fields involved, and cannot repeat the same pattern for each field involved. Moreover, more cards may refer to the same entities involved in the same kind of ontological pattern: what are the instances that can be reused and what are those that cannot?

Returning to our first example on the restoration data, we can notice that the five ICCD fields RSTD, RSTS, RSTE, RSTN and RSTR all refer to the same ontology pattern represented in Fig. 3. This means, for example, that exactly the same instance of the pattern Arch_Find-Restoration, implemented for translating the first field RSTD, must be reused in the translation of all four of the other fields.

Of course, there can be more than one ICCD card, reporting different restorations of the same archaeological find in which the participants remain the same. In this case, the translations of the various restoration data will be made by means of patterns that share the instance of the endurants (**Physical_Person**, **Juridical_Person**, **Archaeological _Find_Person**, etc.) but, for each restoration, each with its own time location at least, different sets of perdurants

(**Restoration** and **Participation)** and different **Time_Region** must be instantiated.

In other words, if in translating RSTD the compiler creates two instances Participation_X1 and Participation_X2, then in translating the RSTS field the same instances Participation_X1 and Participation_X2 must be reused. Viceversa, to record further restorations of the same good, it will be necessary to implement new pairs of instances of Participation.

In order to manage this distinction, the configuration file of ICCD2OWL is provided with global variables that are created when certain precise fields are translated, and that remain visible in all the subsequent processes, until the end of the translation of all the correlated fields.

Trigger field Pattern Name	<global_variables> <field>RSTD</field> <variable_pattern> Restoration_@@RSTN@@ </variable_pattern> <variable_name> RX </variable_name> <condition> <contains> <field>TSK</field> <value>"RA"</value> </contains> </condition> </global_variables>

Table 5. Example of global variable declaration.

When the compiler processes the first field concerning a restoration, let us say RSTD, it creates also an instance of the global variable RX, that contains the id of the instance of the class Restoration. In this way, the information relevant to the status of the translation remains available to the processes that translate the subsequent fields RSTS, RSTE, etc.

An example of global variables codification is given in Table 5.

E. The configuration file

Summarizing, the configuration file of ICCD2OWL is an XML document that contains the codifications of the following items:

- a set of ontological patterns each of which, taken as a whole, represents a not reducible conceptual structure;
- a set of mappings that represents conditional binding of ICCD fields to variables inside the ontological patterns;
- a set of mappings that bind the names used for the classes inside ReMuNaICCD, the symbolic names that represent them inside ICCD2OWL and the rule to create a new instance;
- a set of global variables and the corresponding instructions for use.

V. AN OUTLINE OF THE COMPILER PROCESSES

The formalism introduced above codifies the complete specifications that the compiler must apply in order to translate either ICCD cards into ReMuNaICCD instances or, viceversa, ReMuNaICCD instances into ICCD cards.

A. Compiling ICCD

Until now we have spoken about translating from ICCD to ReMuNaICCD. The following steps are performed as the ICCD file is sequentially read:

1. an ICCD field is read;
2. the appropriate variables are instantiated;
3. the ontological pattern is found that matches the field;
4. the elements of the pattern are processed sequentially; at each step one of the following cases may occur:
   a. the element is a subpattern that contains parameters, then the parameters must be replaced by the actual values;
   b. the element is a triple, then its elements must be instantiated using the current values of the variables;
   c. the element is a class, then the suitable instance of the class is instantiated.

Of course, all the steps of the above process must be performed according to the rules coded in the configuration file previously introduced.

B. Compiling ReMuNaICCD

After the translation, the ontology is transferred into OWLIM [19] a high-performance semantic repository, where the reasoner computes the ontological inferred closure. In order to get both the original and the inferred data in a presentation format familiar to the customer, an OWL to ICCD compiler can be of use.

The process that converts the ontological information to the original ICCD format can be described in the following steps.

For each instance of the ontology, and for each of its properties:

1. the ontological pattern is found that concerns the class of the instance and the property under consideration;
2. the appropriate variables are instantiated;
3. the elements of the pattern are processed sequentially; at each step one of the following cases may occur:
   a. the element is the triple that evaluate variables, then the value of the corresponding property must be passed;
   b. the element is a subpattern that makes references to variables, then the corresponding value must be passed, and the process restart from point 2;
4. the created pattern generates a query, that will be performed on the ontology to retrieve the information about the property.

VI. CONCLUSIONS

Formal ontologies offer a promising approach to facilitate semantic information retrieval from heterogeneous document repositories.

The objective of our ontology was to restore a minimal logical structure to the numerous data that are found in the ICCD cards and whose relation with the original context seems definitively lost.

ReMuNaICCD models a "natural" infrastructure for the accommodation of the data. A first arrangement is determined by the taxonomy of the classes and the taxonomy of the

properties, but the true logic of the model is contained in the patterns of classes and properties that express the group games that the different entities play all together.

Eventually, in trying to populate our ontology with data coming from the available sources, we realized that sensible translations of the traditional data models to ontology models were not at all trivial for two main reasons.

First, translations need pieces of information that could be acquired by the ontology patterns, but these patterns are not explicitly represented inside the RDF/RDFS or OWL ontology codifications.

Second, the translations can be context dependent but none of the current ontology languages is able to express contextual dependence.

Do these reasons mean that the current ontology languages are too poor for solving, without essential auxiliary tools, the problem of semantic interoperability?

In this paper, we investigated the possibility of introducing supplementary formalizations that could be sufficient for the accomplishment of our specific task.

Certainly, the technical details may be improved and the whole formalization may be better situated in the ambits of the most popular standards.

REFERENCES

[1] Pierobon Benoit R., Proto F., Aiello A., Brandi S., Mango Furnari M. 2005, *Concettualizzazione e contestualizzazione dei beni culturali archeologici*, «Archeologia e Calcolatori», 16, 321-339.

[2] de Bruijn J., Polleres A., Lara R., and Fensel D. 2004, *OWL Lite¯*, in J. de Bruijn (ed.), *Final draft d20.1v0.2*, WSML Deliverble

[3] McGuinness D. L. and van Harmelen F. 2004, (eds.), *OWLWeb Ontology Language Overview*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owlfeatures-20040210/.

[4] Volz L. 2000, Web Ontology Reasoning with Logic Databases. PhD thesis, AIFB, Karlsruhe.

[5] Guarino N. and Welty C. 2000, Ontological analysis of taxonomic relationships. In Proceedings of ER-2000: The Conference on Conceptual Modeling.

[6] Guarino N. 1998, Formal Ontology and Information Systems. In N. Guarino (ed.), Formal Ontology and Information Systems, Amsterdam, Nederlands, 3-15

[7] Guarino N. 1995, Formal Ontology, Conceptual Analysis and Knowledge Representation. «International Journal of Human and Computer Studies», 43(5/6), 625-640.

[8] Gangemi A. 2005 Ontology Design Patterns for Semantic Web Content, in E. Motta and Y. Gil (eds.), Proceedings of the Fourth International Semantic Web Conference, LNCS 3729, 262 – 276.

[9] Rector A.L., Rogers J. 2004, Patterns, Properties and Minimizing Commitment: Reconstruction of the GALEN Upper Ontology in OWL, in Core Ontologies Workshop (CORONT), Northampton, UK.

[10] OEP 2004-5, Semantic Web Best Practices and Deployment Working Group, Task Force on Ontology Engineering Patterns. Description of work, archives, W3C Notes and recommendations available from http://www.w3.org/2001/sw/BestPractices/OEP/ (2004-5).

[11] Gennari J., Musen M., Fergerson R., Grosso W., Crubézy M., Eriksson H., Noy N., and Tu S.2003, The evolution of Protégé-2000: An environment for knowledge-based systems development. «International Journal of Human-Computer Studies», 58(1), 89-123.

[12] Knublauch H., Musen M. A., Rector A. L. 2004, Editing Description Logic Ontologies with Protégé OWL Plugin, Proceedings of the 2004 International Workshop on Description Logics, Whistler, British Columbia, Canada.

[13] Broekstra J., Kampman A., van Harmelen F. 2002, Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, in Proc. 1st ISWC.

[14] Giordano M. and Mango Furnari M. 2006 Experimental Ontology Web Services. American Association for Artificial Intelligence.

[15] Aiello A., Mango Furnari M., Massarotti A., Brandi S., Caputo V., and Barone V. 2006, An experimental ontology server for an information grid environment. «International Journal on Parallel Programming».

[16] Patel-Schneider P. F., Hayes P., and Horrocks I. 2004, OWL web ontology language semantics and abstract syntax. W3C Recommendation, 10 Feb. 2004. http://www.w3.org/TR/owl-semantics.

[17] Masolo C., Borgo S., Gangemi A., Guarino N., Oltramari A. and Schneider L. 2003. The WonderWeb Library of Foundational Ontologies and the DOLCE ontology. WonderWeb Deliverable D18, Final Report (vr. 1.0, 31-12-2003)

[18] CIDOC Conceptual Reference Model http://cidoc.ics.forth.gr/

[19] Kiryakov A., Ognyanov D., Manov D. 2005, OWLIM – a Pragmatic Semantic Repository for OWL, in Proc. of International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), Galway, Ireland, November 6-10, 2005, LNCS 3729, pp. 668-684.