Bibliographic Statements

Thomas Severiens

Born in 1970 he studied Physics at the *Carl-von-Ossietzky University Oldenburg*. In 1995 he started to develop *PhysNet*. After graduating, together with some colleagues he founded the *Institute for Science Networking Oldenburg (ISN)*. Since 2004 he is working as a lecturer for *Information Engineering* and as a researcher at the *University of Osnabrück*. He is one of the chairs of *Dublin-Core Working Group "Tools"* and was a member of *W3C Working Group "XQuery"* from 2001 until 2005. His research interests include the development of distributed library systems, user oriented scientific information portals, strategies for longterm preservation of the scientific research output, and development tools for full-text semantics.

Christian Thiemann

Born in 1983 in Osnabrück, Germany. While still in high school, he worked for *Oldenburg Research and Development Institute for Information Technology Tools and Systems (OFFIS)*. After his mandatory military service, he worked for the *Institute for Science Networking Oldenburg (ISN)* in *PhysNet* project (2003-2005). Since fall 2003 he studies Physics at the *Georg-August-University Göttingen*. In 2005/2006 he is studying at the *University of California in San Diego.* Since spring 2005 he also is a member of the working group on *Physics of Transportation and Traffic* at the *University of Duisburg-Essen.* His research interests include cellular automaton models of freeway traffic, computational physics, and computer science in general.

RDF Database for PhysNet and similar Portals

Thomas Severiens, University of Osnabrück, Department for Mathematics and Computer Science, severiens@mathematik.uni-osnabrueck.de

Christian Thiemann, University of California, San Diego, mail@christian-thiemann.de

Introduction

Scientists use the internet to increase the visibility and impact of scholary and scientific outputs, such as publications, research projects, and teaching material. Discipline specific portals like PhysNet[1] can aid scientists in these goals, by covering a specific research field, Physics in this case, and serving the needs of their target group. The underlying motivation of such a portal service is to bring scientists together, presenting the combination of their research interests, and can result in innovative findings. PhysNet is a leading project in Physics information that demonstrates the use of semantic web technologies and distributed user driven services by providing access to information on self archiving publications and about institutional contact data.

We are running and continuously enhancing the PhysNet portal under the auspices of the European Physical Society EPS since 1995. PhysNet is a bouquet of services such as lists of Physics-related institution links (PhysDep), topic specific collections of journals and working groups (PhysTopic) etc. The bouquet contains several search engines on grey literature, open access publications and self archived publications[i].

PhysNet was from its beginning developed as a distributed service[ii], which means that on the one hand the editorial team maintaining the content is distributed all over the world and on the other hand the service itself is mirrored on several hosts to offer an unbiased and highly available service. This results in a thorough coverage, good topicality, and further offered information. PhysNet is a maintained and open environment for new developers and team members to set up additional mirroring servers, and for further patronizing societies. The basic principles can be found in the charter[2].

This article focuses on the data model being used to store the knowledge of the portal and on the used techniques to allow a distributed maintenance of this kind of content.

Separation of Data and View

To offer additional, more interactive and sophisticated services and interfaces to PhysNet, it was requested to separate the knowledge from its representation. Having already information on over 1,000 institutions in the system, this was no triviality.

Until late summer 2004 PhysNet was a huge collection of HTML files containing the actual data, i.e. institution names, their locations and homepage and publication list URLs, embedded within the view, i.e. the HTML markup. The coarser geographical structure (continents, countries) was given by the links between the HTML files, the finer geographical structure (states, cities) was stored in HTML description lists as well as the institutional structure. While a human user could easily identify cities in e.g. PhysDep service due to the bold printing and recognize the structure of a physics department's institutes due to the different indention, a machine had to infer this information in the same way by interpreting the visualization of the data instead of having precise metadata. Furthermore, small typing errors made while maintaining the lists destroyed even this weak metadata.

Therefore, and to be able to extend PhysNet with other services than just providing URLs, in October 2003 we started examining the structure of the PhysNet data in order to extract it into a database with clear metadata. As one can easily recognize by browsing through PhysDep or PhysDoc, the PhysNet data shows a very simple structure. Starting at the root node "PhysNet World" one has to select a continent and a country until a list of cities is displayed. Each institution is either associated with

[1] PhysNet: www.physnet.net
[2] PhysNet charter: www.physnet.de/PhysNet/charter.html

another institution or with a city or country etc. This identifies the PhysNet data o be a tree structure, thus XML seems to be the most appropriate data model to use. But a more detailed look shows that certain nodes do have more than one parent. For example, Turkey is counted as a country of Europe as well as a country of Asia, or the "University of California in San Diego" is associated with the city of San Diego but also with the institution "University of California" which is associated with the state California. Therefore, the PhysNet data is actually a graph for which RDF[iii] is the perfect data model. RDF allows for easy storage by using serialized RDF triples in MySQL databases while RDF Schema combined with OWL Lite provides an adequate language for defining the metadata. On the other hand, while XML allows for editing the data by just using a simple text editor, the RDF graph is more complex, which is the reason that one has to use a more sophisticated editor in order to be able to manage the data amounts and structures as we have to handle with in PhysNet.

The decision for RDF triples as the data model was motivated by a number of reasons:

- The data model should be able to encode all implicitly encoded information of the existing HTML files of PhysNet and combine those parts of information, which was redundantly typed into the files (e.g. sub-institutions of institutions located in more than one city). The fact, that the information is a graph, not a tree of hierarchical information, gave a first hint on RDF.
- The data model should be open for any kind of future developments. Semantic encoding in a triples collection and definition of an ontology is perfectly fitting this requirement as it only may request new predicates for the triples to be defined in the ontology.
- It should be possible to mirror and run the database at selected mirror sites. This asked for development of a database structure, which is not essentially requesting a specific implementation of a database software. Our implementation uses only one table with three columns for the data and one other table of the same structure for the metadata.

We were aware of the disadvantages this solutions has:

- Due to the fact that the triples are unsorted, operations on the database are slow by default, even if for most of the requested operations quick solutions could be found. To offer an acceptable performance to those users operating on the database directly it is essential to hold it in the main memory of the server. Due to the effect, that nearly all interaction with public users are running on static HTML files (views) generated every night, the resulting load of the server is low.
- Offering more interactive views like WebServices or XQuery based networking, it could be requested to transfer the knowledge into a native RDF or RDF-XML database to increase the speed of the interface. In 2004, when a decision on the architecture had to be made, RDF databases were still in early development stage and too unstable, from our point of view, to trust them. We wanted to make sure that we will always be able to change or expand the data model and technology.

The PhysNet Ontology

In the following we will describe the metadata we created for storing the data presented in PhysNet. The ontology defining the predicates in the triples follows the conventions of RDF Schema[iv] and OWL Lite[v], but the related RDF-editor recognizes a few semantic extensions which will be introduced in this section and specified later on. This part will be pretty formal and detailed but only by that the reader may be able to set up related services of his own.

Figure 1 shows a diagram representing the class hierarchy in the ontology. Each arrow pointing from class A to class B states that class A is a subclass of class B, thus inherits all properties of class B. However, to improve readability, some simplifications had been made: We omitted the namespace prefixes, as we will do throughout most of this article, and the class `physnet:Object`, which is just a simple (abstract) base class without any properties that is used to make the hierarchy graph a tree. All classes that seem to have no superclasses (base classes) are actually a subclass of `physnet:Object`. Classes which names are printed in italic are defined with the namespace prefix `physnet_abs` while all other classes are defined with `physnet` except for `PACSCode` which is

defined with the namespace prefix `pacs` (for Physics and Astronomy Classification System[3]). Similarly, all properties tagged by `[abs]` are defined with `physnet_abs` while all others are defined with `physnet` except for `parent` in `pacs:PACSCode` which belongs to the namespace `pacs`.

[3] Physics and Astronomy Classification System: publish.aps.org/PACS/

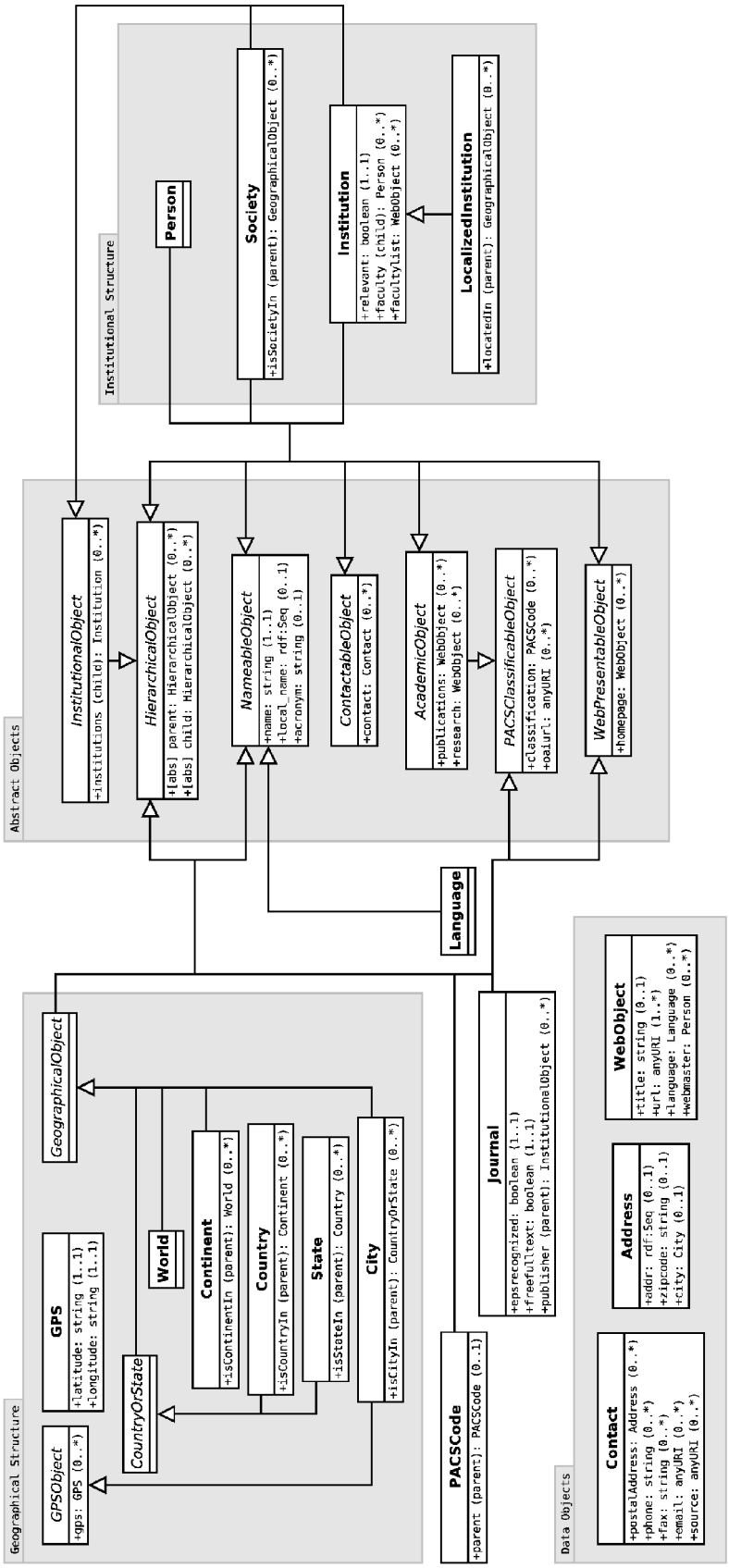

Figure 1: Class hierarchy in the PhysNet ontology.

All classes and properties defined with `physnet_abs` are considered to be abstract. The concept of abstract classes is an extension to RDF Schema and OWL Lite. The RDFEditor will not offer an option to create an instance of such a class nor offer to assign a value to such a property. While the concept of class inheritance and abstract classes is familiar from other object-orientated

environments, the concept of property inheritance and abstract properties might be new to the reader. In the diagram, we list the properties of each class under the class name. In each line, the name of the property, its range (type) and cardinality is given. Since we use OWL Lite, the only cardinality restrictions available are “unlimited” (`0..*`), "max. 1” (`0..1`), “at least 1” (`1..*`), and “exactly 1” (`1..1`). If a property is a subproperty, the name of its superproperty is given in parentheses before the colon.

As already mentioned, the PhysNet data contains a lot of hierarchical structure. To represent this in the editor and to simplify navigating through the data, we introduced the abstract class `physnet_abs:HierarchicalObject` with the two abstract properties `physnet_abs:parent` and `physnet_abs:child`. Both properties have the range `HierarchicalObject` and any assignment of a value “Instance B” to `parent` (`child`) on “Instance A” states that “Instance B” is considered to be a parent (child) of “Instance A” in the logical hierarchy. The RDFEditor uses these special semantics to provide a display of the hierarchical neighborhood around one particular instance instead of listing all instances of a certain type or other matching criteria and thus drastically simplifies navigation inside the PhysNet data graph.

Another semantic extension of the RDFEditor is the `physnet_abs:NameableObject` which provides the properties `physnet:name`, `local_name` and `acronym`. This class is meant to be a base class representing any type of object that can have a name. The RDFEditor recognizes such instances and displays them using their name rather than their RDF-URI. Note that in contrast to `HierarchicalObject` the properties of `NameableObject` are non-abstract. While `name`, `local_name` and `acronym` will always have simple strings as values, the different types of objects in the hierarchy might be restricted to different parent or child objects which requires them to define a new property derived from `parent` or `child` but further restrict the range of that property.

The simplest example of using the `HierarchicalObject` and the `NameableObject` is representing the PACS tree by using one instance of `PACSCode` for each code. `PACSCode` defines a new property `pacs:parent` derived from `physnet_abs:parent` that has `PACSCode` as range. The PACS code and text is stored in the `name` property. This design results in a subgraph of `PACSCode` instances in the PhysNet data graph. Practically, since we represent the official PACS tree, this subgraph is of course a tree.

Another class derived from those two abstract classes is `GeographicalObject` which serves as a base class for any object representing geographical structure, in particular `World`, `Continent`, `Country`, `State`, and `City`. Each one of them defines a subproperty of `parent` that states the containment of one geographical instance in another. `Country` and `State` are not direct subclasses of `GeographicalObject` but are derived from `CountryOrState` because we needed cities to be located in countries as well as in states, but there is no way in RDF Schema to specify that values on a specific property might be of either type A or type B. `City` is also a subclass of the abstract `GPSObject` which adds the `gps` property to all instances of `City` to which one can assign a `GPS` instance containing the longitude and latitude of the city. While GPS data was first gathered for cities only, currently the exact positions of single institutions are collected. Using this information PhysNet will be able to calculate the distances between cities and offer searches like “Show all institutions 50 km around city A or institute B independent of borders of countries or continents”. Currently we are evaluating whether this kind of service should be implemented on the database level, many databases already implement these algorithms, or on the service level, which would allow us to stay independent of the selected database software.

As shown in the lower left corner of the diagram, we defined three pure data objects representing contact information, postal addresses and metadata about websites. `Address` is a simple structure that can be used to represent a complete postal address. The first address lines containing recipient and street name are stored using an RDF Sequence container. The zipcode is stored in a simple string and the `city` property refers to an instance of `City` in the PhysNet data. Therefore the country can be inferred by following the `isCityIn` property of the value on `city`. The `Contact` class provides properties to store postal addresses and other contacts like email and phone. The `source` property is not an actual data field shown on PhysNet but is used to specify the URL of the website(s) from which the contact information was taken. A data validation script, that continuously checks for broken links in the PhysNet data, uses this value to verify that the contact information is still up to

date. The WebObject is used to represent the content, PhysNet was launched for: web links. Each instance must have a value on url giving the actual URL and may optionally provide further information on title, language and webmaster.

All objects that might have a contact or a homepage have to be an instance of the abstract class ContactableObject or WebPresentableObject which provide the two properties contact and homepage. Another class referring to WebObject is the abstract class AcademicObject defining the two properties publications and research which are meant to store the links provided in PhysDoc, while the information given on the property homepage is used in the PhysDep lists. Every AcademicObject is also a PACSClassificableObject. This abstract class was introduced to add the property classification to Institution in order to be able to associate PACS codes with institutions. It evolved to a base class for any object that can be associated with any astronomy or physics related field and now provides also the oaiurl property, where URLs of OAi data-providers[4] are stored.

Introducing the abstract class InstitutionalObject leads us to the description of the institutional structure in the PhysNet data graph. Each institution like universities, departments, research groups or independent research institutes is represented by an instance of Institution and due to the institutions property inherited from InstitutionalObject each such institution can have "child institutes". Using this property, departments can be associated with universities and research groups with departments. Different universities have different types of institutional structure, thus we did not add any more specific meaning to the Institution object and left the recognition of the type of the actual instances to the PhysNet user. Therefore there is no technical difference between e.g. the "University of Göttingen", the "Physics Department" and the "Institute for Theoretical Physics" although the structure stored in the PhysNet data graph and shown together with the institution's name in PhysDep or PhysDoc implies significant differences between the three institutions. However, Institution provides the mandatory relevant property whose value is used to evaluate whether an institution is doing mainly Physics related research or not. Thus, in the above example, "University of Göttingen" would have a false value on relevant while the other two would have a true value. This implementation allows field specific robots to extract the relevant starting URLs for building up field specific learned search engines.

The institutions represented by Institution are actually interpreted to be some sort of child institutions located in a bigger institution. For universities or independent research institutes we use instances of LocalizedInstitution that adds the locatedIn property to specify a GeographicalObject in which this institution is located. This property, as a subproperty of HierarchicalObject's parent, connects the institutional structure graph with the geographical structure graph.

Another element of the institutional structure is the Person class used to represent specific persons that may have a homepage or publication list. Persons can be inserted into the PhysNet hierarchy by adding them as values to the faculty property of some Institution instance. Furthermore, Person instances can be used to store contact information about a webmaster of a website using the webmaster property of WebObject. The validation script that checks for broken links uses the email value of the Contact instance on the Contact property of the Person instance on the webmaster property of the instance of the WebObject storing the broken link, to send a predefined email asking for the new URL of the website.

The Society class is the last element of the institutional structure and is used to represent physical societies. The property isSocietyIn connects to the geographical structure and is used to specify the country or continent the society is associated with. Technically there is no difference between an instance of Society and LocalizedInstitution in the PhysNet data, but instances of Society are also displayed in the Societies listing of the PhysNet service while instances of LocalizedInstitution are shown in PhysDep and PhysDoc only. Even semantically there is sometimes little difference between societies and what we understand as localized institutions, as there are some societies like the "Max Planck Society" that supervise several research institutes but there are also institutes like the Italian "Istituto Nazionale di Fisica Nucleare (INFN)" which are split

[4] Open Archives Initiative: www.openarchives.org

up into several small research institutes spread out over a whole country and may be seen as a society as well.

Finally, to represent journals as shown in the PhysNet's Journals list, we defined a class `Journal`, that adds the boolean properties `epsrecognized` and `freefulltext` to the inherited properties from `NameableObject`, `WebPresentableObject` and `PACSClassificableObject`. These two properties are used to generat the lists of only EPS recognized journals and journals with free fulltext available. Since many journals are published by physical societies or institutes, we also derived Journal from `HierarchicalObject` and introduced the `publisher` property, which connects to the institutional structure.

RDF Online-Editor

As mentioned above, we need an adequate editor to manage the data stored in the RDF graph. During the design phase of the editor we were thinking whether it would be better to build an editor which exactly fits the structure of the PhysNet data or offers the full flexibility of RDF. We decided to go for more flexibility and created RDFEditor being able to edit any RDF graph described by normative RDF Schema and OWL Lite while still offering convenient data entry forms due to support plugins. The plugins offer a convenient method to initiate a whole set of operations on the RDF graph.

As described in the preceding section, the webmaster's email address for some webpage is stored in a `Contact` instance associated with a `Person` which is associated with the respective `WebObject`. Adding a new link with webmaster email address in pure RDF is a tedious task of creating three RDF objects and linking them together correctly. That is the way the RDFEditor offers the full flexibility of RDF to the user, but in this case it was more than convenient to create a plugin that displays a simple form with input fields for URL and email address and takes care of the creation and linking of the RDF objects.

The same plugin, called PhysNetToolbox, offers a form for entering contact information including postal address, thus combining the creation of a `Contact` and `Address` instance in one step. However, the price for simplicity is the loss of flexibility as with using only the PhysNetToolbox there is no possibility to create more than one postal address per contact or to insert more than three name and street lines to a postal address. But then, one can use the basic RDF editing capabilities of RDFEditor to enter the data as needed. This combination of being able to handle RDF on its basic level and the support for user-defined plugins makes RDFEditor a very powerful tool.

Furthermore we had to decide on the platform on which to implement the editor. Since some countries in PhysDep and PhysDoc are not maintained by us but by volunteers from all over the world, we needed a system that allows editing the PhysNet data from various places. We decided to make the RDFEditor completely web-based so that the only thing one needs to use it is a web browser.

RDFEditor can handle multiple users and for each of them certain editing restrictions can be enforced. It is possible to create an account for somebody who is responsible for the data of the e.g. German PhysDep list and who can only edit RDF objects that are children (in the sense of the `child` and `parent` properties of `HierarchicalObject`) of the `Country` instance "Germany".

The Data Model

The data model that is processed by RDFEditor can in theory be any RDF graph. It requires the graph to be serialized into RDF triples and stored in a MySQL database. However, the internal design of the RDFEditor allows for other input/output modules such as XML import or export.

The data model has to be described using OWL Lite which must be provided as an RDF graph also stored in a MySQL database. This description is interpreted to be normative, thus RDFEditor does not allow for the creation of classes that are not defined in the metadata. Almost every feature of RDF Schema and the cardinality restrictions of OWL Lite are supported, the only exception is that RDFEditor does not allow for resources that are instances of more than one class. This, however, enforces object-orientated design of the metadata. Furthermore, RDFEditor implements some additional semantic features that are described in the following.

 # Definition of a World
'physnet:world','rdf:type','physnet:World'
'physnet:world','physnet:name','PhysNet World'
 # Definition of Europe
'physnet:108098549397184689428 1','rdf:type','physnet:Continent'
'physnet:108098549397184689428 1','physnet:name','Europe'
'physnet:108098549397184689428 1','physnet:isContinentIn','physnet:world'
 # Definition of Asia
'physnet:108098549291174679853 3','rdf:type','physnet:Continent'
'physnet:108098549291174679853 3','physnet:name','Asia'
'physnet:108098549291174679853 3','physnet:isContinentIn','physnet:world'
 # Definition of Turkey
'physnet:109108474884939277521','physnet:name','Turkey'
'physnet:109108474884939277521','physnet:local_name','_:109108474884238966480'
'physnet:109108474884939277521','rdf:type','physnet:Country'
'_:109108474884238966480','rdf:type','rdf:Seq'
'_:109108474884238966480','_:1','Türkiye'
 # Location of Turkey
'physnet:109108474884939277521','physnet:isCountryIn','physnet:108098549291174679853 3'
'physnet:109108474884939277521','physnet:isCountryIn','physnet:108098549397184689428 1'

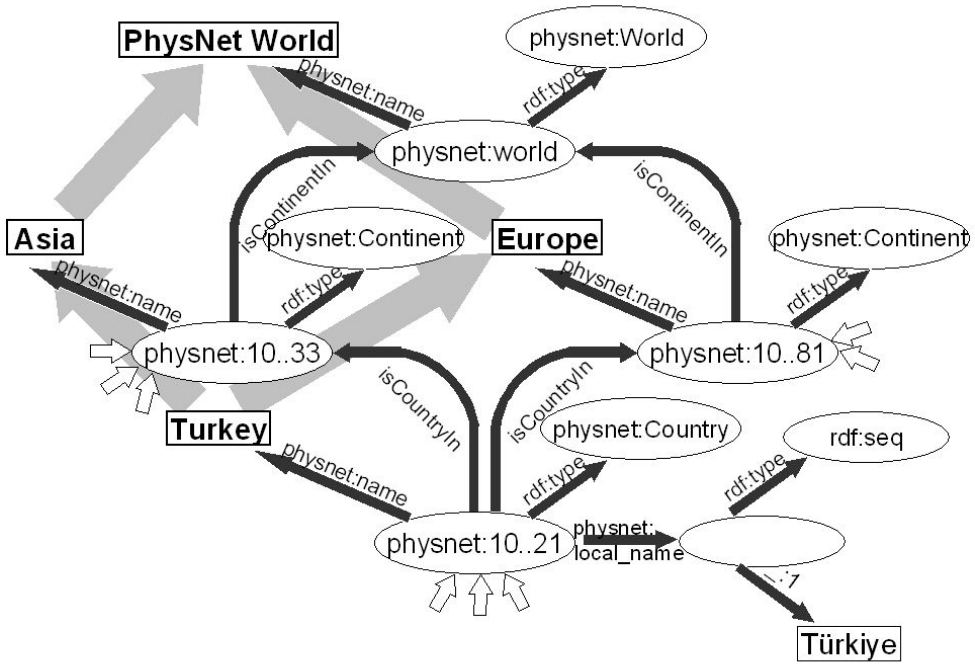

Figure 2: RDF triples and graph to describe that Turkey is located in Europe and Asia, which are continents of the world, and that Turkey has a local name "Türkiye".

Abstract Classes and Properties: In general, any of the classes defined in the metadata can be created by using the RDFEditor and values can be assigned to any properties defined in the metadata. Since this is sometimes not intended, the RDFEditor offers to specify an abstract namespace prefix. All classes that are defined using that namespace prefix will not be shown in the RDFEditor's class list as well as properties defined using that namespace prefix will be hidden in the RDFEditor's resource edit panes.

Instance Names: RDFEditor allows for specifying the name of a class and one of its properties. All instances of this class are then considered to have a name given as the (only) value of the specified property. This name is used to represent the instance in listings or titles etc. If no such class is specified, all RDF instances are listed using their URI which often is a cryptic value.

Hierarchy: Data stored in RDF graphs often show some sort of hierarchy, which can be exploited to simplify navigation in the graph. In the configuration of RDFEditor the name of a class and two of its properties can be given. All instances of this class are then interpreted as nodes in the hierarchy, while values on the two properties are interpreted as pointers to parents or children of a particular instance. With this information given, RDFEditor is able to replace the generic resource list, which simply lists all instances in the data model, with a hierarchical resource list which lists parents and children of one particular instance and allows for traversing through the graph along the hierarchy edges (`parent`/`child` properties).

Special Datatype Interpretation: The RDFEditor offers one simple HTML text input for every value assigned to a property. However, sometimes it is more convenient or necessary to provide different input fields. If the range of some property is `xsd:boolean`, values on that property are shown as a two-radio-button group rather than a text input field where one would have to write either "true" or "false". If the range of some property is `rdf:password`, a HTML password input field is shown to hide the content from the screen and the contents of this field are stored MD5 encoded instead of plain text.

The User Interface

The RDFEditor's web based user interface is modular. Every browser window is considered to be a *frame* containing several user interface components, which are presented in the following. Throughout, the RDF term *type* is used instead of the object-orientated term *class*, and *resource* is used instead of *instance*.

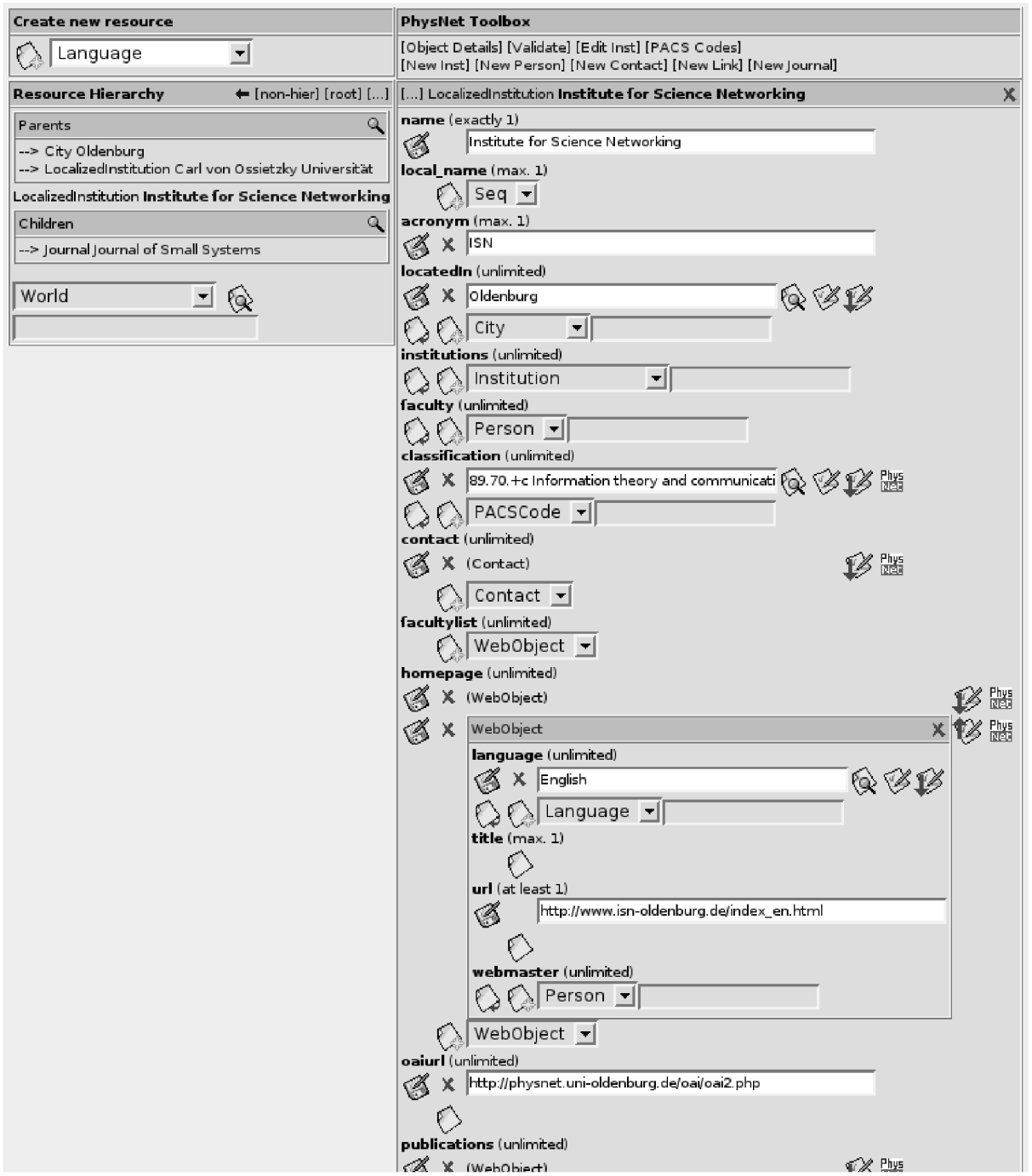

Figure 3: Screenshot of RDFEditor currently editing the resource "Institute for Science Networking"

Generic and Hierarchical Resource List: The generic resource list simply lists all resources which exist in the currently edited data model and match certain filter criteria. A variant of the resource list is the hierarchical resource list (fig. 4) which shows parents and children of a specific resource. The relations "parent" and "child" must be defined in the metadata as explained above.

Figure 4: Hierarchical Resource List

For each resource its type and its URIref or name is displayed. The URIref (or name) is a link which performs an action depending on the context of the resource list. The resource list in the main editor frame (fig. 3) sets the currently edited resource in the resource edit pane to the resource which URIref (or name) has been clicked.

The hierarchical resource list additionally shows an arrow ("`-->`") before each resource. By clicking this link you can follow the hierarchy, i.e. the resource which parents and children are shown is set to the resource which arrow has been clicked. On the right side of the hierarchical resource list a "Change resource" button is presented (`[...]`) which asks the user for an URIref and sets the resource list's resource to the one specified by the user. Next to it the hierarchical resource list offers a button to switch to a generic resource list ("`[non-hier]`" in the title row). The resource list which is shown after pressing this button offers a button to switch back to the hierarchical resource list.

The list of resources shown by a resource list can be restricted using a filter. By clicking the magnifying glass in the title bar one can open the resource list's filter setting dialog (fig. 5). The RDFEditor distinguishes two different kinds of filters: type filters and property value filters.

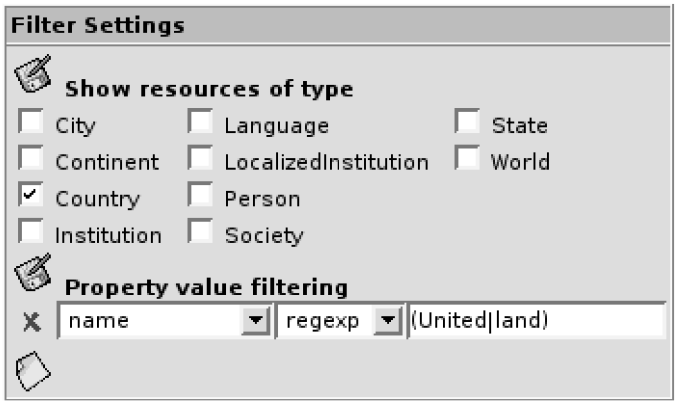

Figure 5: Filter setting dialog

The type filter is set in the upper half of the filter settings dialog. The resource list will display only those resources the type of which is among the selected ones.

The property value filters are configured in the lower half of the filter settings dialog. By clicking the "New property value filter" button a new filter can be created. Existing property filters can be deleted by clicking the "Delete property value filter" button which is shown left to each filter. For each filter three input elements are displayed: The left drop-down box shows a list of properties defined in the metadata. The right drop-down box offers "regexp" and "equal".

If "regexp" is selected in the right drop-down box, the value given in the text field is interpreted as a regular expression and the filter rejects all resources which do not have at least one value on the selected property that matches the regular expression. If "equal" is selected the filter rejects all

resources which do not have at least one value on the selected property which exactly matches the string entered in the text field. The comparisons are case-sensitive.

The filter configured in figure 5 matches all resources of type “Country” which have a value on the property “name” that contains either the string “United” or “land”.

Resource edit pane: The resource edit pane (fig. 6) is the facility for editing a resource. The type and the identity of the resource are shown in the title bar (the name of the resource is displayed instead of its URIref if the editor is configured for nameable objects). By clicking the red cross in the title bar the whole resource can be deleted from the RDF data. This results in the erasure of all triples which subject or object matches exactly the URIref of the shown resource.

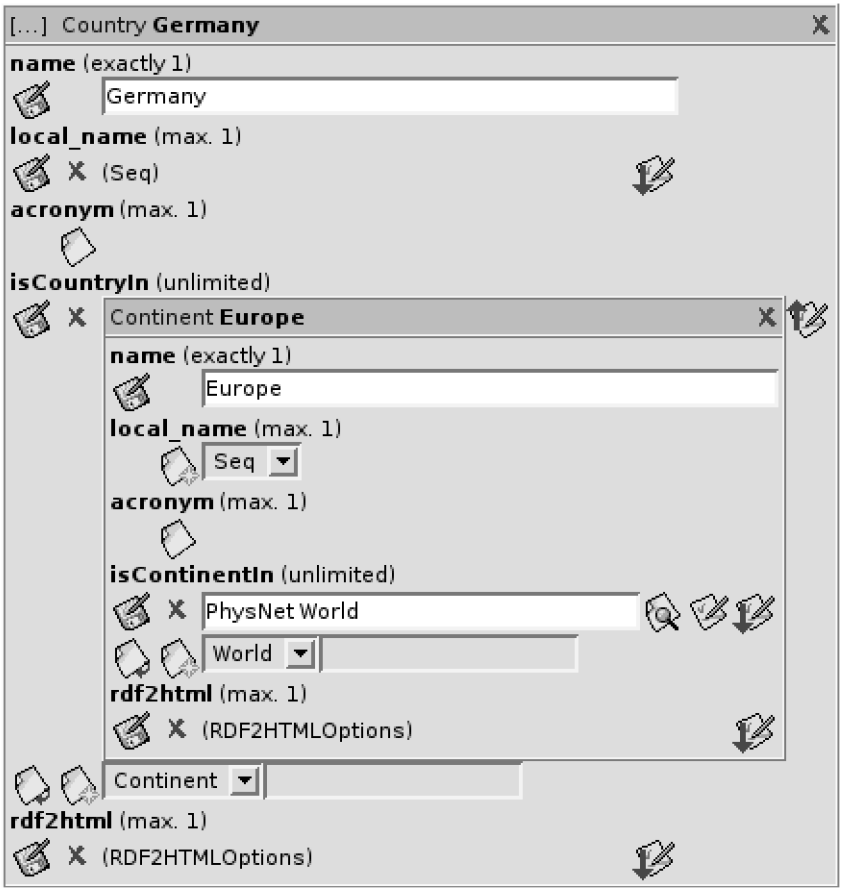

Figure 6: Resource edit pane

The type of a resource can be changed by clicking its type in the title bar. However, this feature is to be used with caution because it may result in non-metadata-conform data since it only changes the type of the resource regardless to whether the properties values are stored for in the data are still defined with the new type.

Left to the resource edit pane’s title is a “Change resource” button (`[...]`) which, similarly to the hierarchical resource list’s equivalent, asks the user for an URIref and changes the edited resource to the one specified by the user.

The pane lists all properties which can be applied to the resource and the values on that properties. If a value is a literal value, a text field is displayed. If a value is the URIref of another resource, i.e. a reference, the pane may show a sub-resource-edit-pane showing the referenced resource.

Next to the property name the cardinality is displayed. Possible values are “unlimited”, “max. 1”, “at least 1” and “exactly 1”.

If the resource is not editable by the currently logged in user, the resource edit pane shows only the plain data without any edit options. That means, most of the buttons are not shown.

Container edit pane: The container edit pane (fig. 7) is a specialized resource edit pane which is designed for editing `rdfs:Containers`. In general, it behaves like a normal property except that the values are numbered. The buttons to the left of the input fields insert or delete values into/from the list (the "Insert" button creates a new value on the current index, all values next to and below the button are shifted one index number). The buttons to the right change the order of the values.

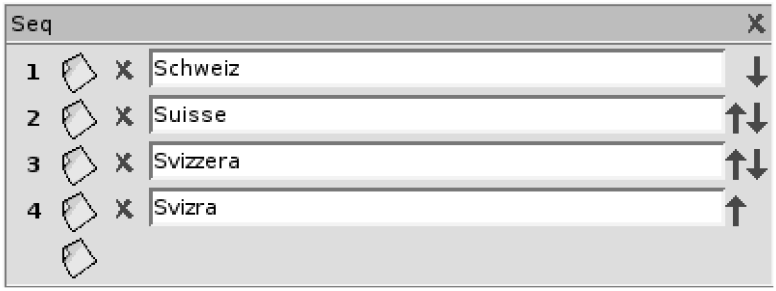

Figure 7: Container edit pane

User Management

As mentioned above, RDFEditor supports multiple users. Access to data models and even single resources or subgraphs of a data model can be denied or granted for each user.

Figure 8 shows an example user which has access to the data model "myDatamodel" but is not allowed to edit resources below the resource "mydata:world". In the example data model the resource "mydata:germany" is a child of "mydata:europe" which is a child of "mydata:world". Eddie is allowed to edit "mydata:germany"because it is explicitly mentioned in "allowedResources". Also the whole subtree below "mydata:germany" (which contains for example "mydata:berlin") can be edited by Eddie while "mydata:europe" and other subtrees below "physnet:world" (for example "mydata:asia") will appear read-only to Eddie.

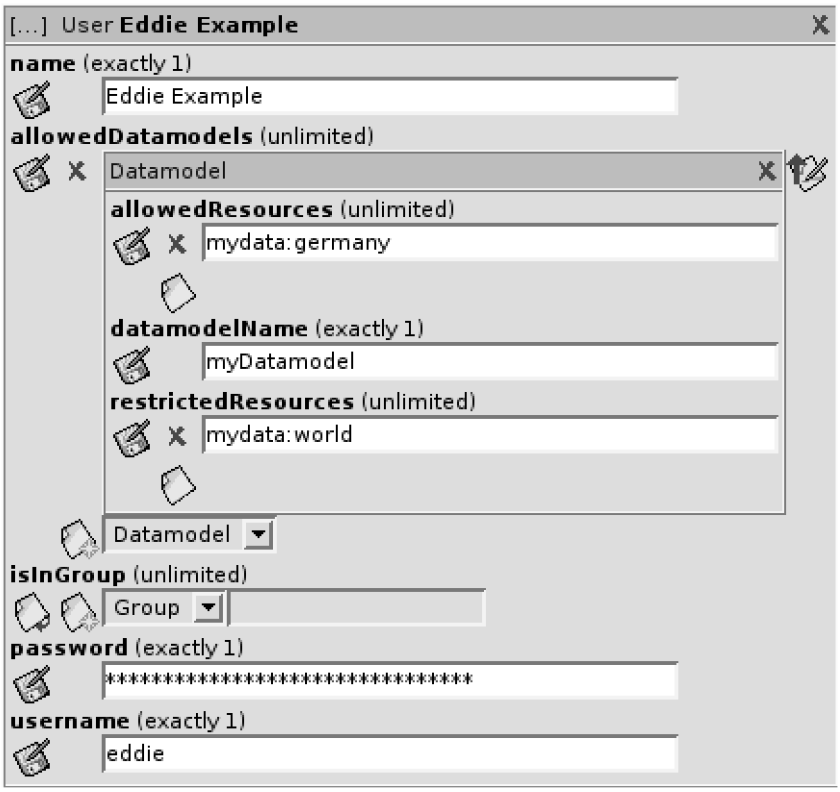

Figure 8: User with restricted write access

Outview and Next Steps

In 2005 and 2006 we successfully transferred the knowledge of PhysNet from static HTML files into the database of RDF triples. Meanwhile, the knowledge within the database is continuously being updated and enriched. A semi-automatic tool was developed, which checks for broken links and outdated information like phone numbers and other fragile information e.g. in the `ContactableObjects`.

For summer 2006 we are planning a tutorial workshop with the goal to increase the number of editors and further optimize the RDFEditor tool. This is desirable because topic related services require a granularity of the institution on at least working group level.

One problem occurs to be serious but independent of the data model: PACS classification is frequently updated as a result of upcoming new fields and research topics in Physics. A unique migration path is not always given, resulting in a high amount of intellectual work being left after automatic migration to the latest version of PACS. Currently PhysNet is still using PACS 2003 instead of the latest 2006 version of the classification.

Networking with other portals is currently one of the most relevant topics. As an example, let us have a view into ViFaPhys[5], the Physics Virtual Library. Their collection of web resources, of PACS classified online and offline journals would perfectly fit into a topic specific portal service like PhysTopics. On the other hand, their service lacks a well-maintained list of institutions, like PhysNet offers.

Networking with similar portals from other research fields may result into new and innovative services. Sharing common parts of information, like the geographical information (independently of the field the continents are equal, even the majority of institutions, their postal address, their GPS coordinates etc. are identical), will increase interdisciplinary communication and make use of the available synergies in maintaining online information services for the learned fields. WebServices using XML SOAP and XQuery are promising candidates for the requested networking infrastructure.

Acknowledgements

A major part of this work has been done at the Institute for Science Networking ISN[6] with support of the European Physical Society EPS[7].

Bibliography

[5] ViFaPhys the Physics Virtual Library: www.vifaphys.de
[6] Institute for Science Networking ISN: www.isn-oldenburg.de
[7] European Physical Society EPS: www.eps.org

[i] T. Severiens, M. Hohlfeld, K. Zimmermann, E. R. Hilf: *PhysDoc - A Distributed Network of Physics Institutions Documents - Collecting, Indexing, and Searching High Quality Documents by using Harvest* D-Lib Magazine, Vol. 6 No. 12, December 2000

[ii] E. R. Hilf, M. Hohlfeld, T. Severiens, K. Zimmermann: *Distributed Information Services in Physics* in High Energy Physics Libraries Webzine [ISSN 1424-2729] Issue 4 / June 2001

[iii] F. Manola, E. Miller, B. McBride: *RDF Primer*, W3C Working Draft, October 10, 2003, http://www.w3.org/TR/rdf-primer/

[iv] D. Brickley, R.V. Guha: *Resource Description Framework (RDF) Schema Specification*, World Wide Web Consortium, March 27, 2000. http://www.w3.org/TR/2000/CR-rdfschema-20000327/

[v] M. Dean, G. Schreiber, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L.A. Stein: *OWL Web Ontology Language Reference*, World Wide Web Consortium, March 31, 2003 (work in progress), http://www.w3.org/TR/owl-ref/