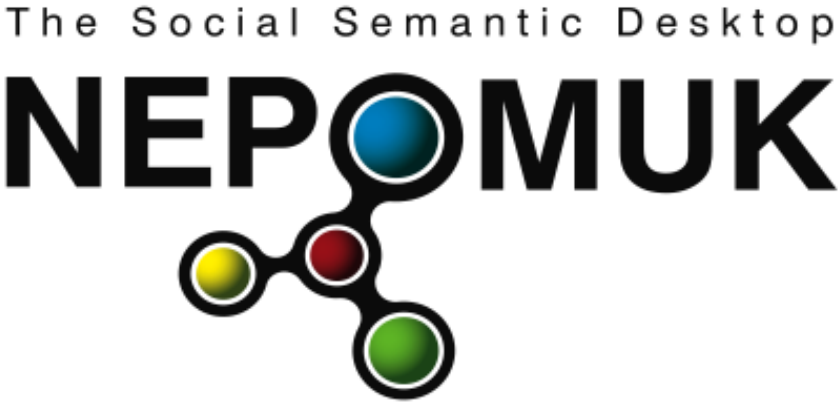
Integrated Project

Priority 2.4.7

Semantic based knowledge systems

The Social Semantic Desktop



Personal Information Model (PIMO)

NEPOMUK Recommendation v1.0

Version 1.0
02.09.2008
Dissemination level: PU

Nature	Report
Lead contractor	DFKI
Start date of project	01.01.2006
Duration	36 months

Authors

Leo Sauermann, DFKI
Ludger Van Elst, DFKI
Knud Möller, DERI

Project Co-ordinator

Dr. Ansgar Bernardi
German Research Center for Artificial Intelligence (DFKI) GmbH
Trippstadter Strasse 122
D 67663 Kaiserslautern
Germany
Email: bernardi@dfki.uni-kl.de, phone: +49 631 205 3582, fax: +49 631 205 4910

Partners

DEUTSCHES FORSCHUNGSZENTRUM F. KUENSTLICHE INTELLIGENZ GMBH
IBM IRELAND PRODUCT DISTRIBUTION LIMITED
SAP AG
HEWLETT PACKARD GALWAY LTD
THALES S.A.
PRC GROUP - THE MANAGEMENT HOUSE S.A.
EDGE-IT S.A.R.L
COGNIUM SYSTEMS S.A.
NATIONAL UNIVERSITY OF IRELAND, GALWAY
ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
FORSCHUNGSZENTRUM INFORMATIK AN DER UNIVERSITAET KARLSRUHE
UNIVERSITAET HANNOVER
INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS
KUNGLIGA TEKNISKA HOEGSKOLAN
UNIVERSITA DELLA SVIZZERA ITALIANA
IRION MANAGEMENT CONSULTING GMBH

Versions

Version	Date	Reason
0.1	01.06.2007	a template of the document prepared by Antoni Mylka
0.9	12.08.2008	Finished for review.
1.0	02.09.2008	Reviewed and accepted.

Explanations of abbreviations on front page

Nature
R: Report
P: Prototype
R/P: Report and Prototype
O: Other

Dissemination level
PU: Public
PP: Restricted to other FP6 participants
RE: Restricted to specified group
CO: Confidential, only for NEPOMUK partners

Table of contents

1 Abstract

The PIMO Ontology can be used to express Personal Information Models of individuals. It is based on RDF and NRL, the NEPOMUK Representational Language and other Semantic Web ontologies. This document describes the principle elements of the language and how to use them.

2 Status of this document

This section describes the status of this document at the time of its publication. The form used for this status message and document is inspired by the W3C process.

This document is an NEPOMUK recommendation produced by Leo Sauermann and Knud Möller as part of the task-force Ontologies in the NEPOMUK Project. The document has been promoted from a draft form to this official form upon reviewing by the general NEPOMUK consortium. Subsequent versions of PIMO might mean that the specification documents of the later versions render this document obsolete, with respect to the version of PIMO in use, but not with respect to this version.

This document and the PIMO ontology as such is a continuation and improvement of existing work. Other documents may supersede this document. Parts of this document will be published in other documents, such as scientific publications. This document is based on various other publications by the authors, and is a continuation of existing work. Some formulations from the RDFS primer and SKOS primer documents were reused.

Additional to this document, a FAQ is maintained in the public NEPOMUK wiki[1]. **This document is accompanied by a RDFS/NRL ontology[2], which should be downloaded (see Section 3.1) and read in parallel to learn more about PIMO.**

The editors of this document value feedback from from the public and from the NEPOMUK consortium. Typographic errors and change requests of the ontology can be reported using the NEPOMUK tracker[3], using the category **ontology-pimo**. General questions about using the ontology will be answered in the FAQ. Subsequent versions of the ontology will include improvements gathered in this way.

We want to thank our reviewers for their feedback: Ansgar Bernardi, Siegfried Handschuh, Pär Lannerö, Enrico Minack, Gerald Reif, and Max Völkel.

3 Introduction

PIMO is the abbreviation for the *Personal Information Model* of a user. The PIMO-Ontology is both an RDF vocabulary[4] to express such a model and an upper ontology defining basic classes and properties to use.

Readers of this document should be familiar with the Resource Description Framework on the level described in the RDFS-Primer [2] and the NRL specification[5]. The emphasized key words "*must*", "*must not*", "*required*", "*shall*", "*shall not*", "*should*", "*should not*", "*recommended*", "*may*", and "*optional*" in

[1] `http://dev.nepomuk.semanticdesktop.org/wiki/PimoFaq`

[2] `http://www.semanticdesktop.org/ontologies/2007/11/01/pimo#`

[3] `https://dev.nepomuk.semanticdesktop.org/newticket`

[4] More precisely, PIMO is based on the NEPOMUK Representational Language (NRL), which extends the RDF model with a semantic model and support for named graphs and other features. However, for the sake of simplicity, we will use the term RDF throughout most of this document.

[5] `http://www.semanticdesktop.org/ontologies/nrl/`

this document are to be interpreted as described in RFC 2119.

The scope of a PIMO is to model data that is within the attention of the user and needed for knowledge work or private use. The focus is on data that is accessed through a Semantic Desktop or other personalized Semantic Web applications. We call this the **Personal Knowledge Workspace** [9] or **Personal Space of Information (PSI)** [10], embracing all data "*needed by an individual to perform knowledge work*". It is (1) independent from the way the user accesses the data, (2) independent from the source, format, and author of the data. The abbreviation **PSI** will be used.

Today, such data is typically stored in files, in Personal Information Management (PIM) or in groupware systems. A user has to cope with different formats of data, such as text documents, contact information, e-mails, appointments, task lists, project plans, or an Enterprise Resource Planning (ERP) system. Existing information that is already stored in information systems is in the scope of PIMO, but abstract concepts (such as "Love", "Language") can also be represented, if needed.

PIMO is based on the idea that users have a *mental model* to categorize their environment. It represents the user itself and the fact that he has a *Personal Information Model*, this is described in Section 6.1. Building a PIMO for the imaginary user *Claudia Stern* is the guiding example for this document. Each concept in the environment of the user is represented as *Thing* in the model, and mapped to documents and other entities that mention the concept (see Section 6.2). Things can be described via their relations to other Things or by literal RDF properties, the key properties are defined in Section 6.14 and directions given how to use them. An important question is the class of a Thing; the semantics of classes and top level classes are presented in Sections 6.11-6.13. To match and align similar concepts, existing identification systems are reused and integrated (Section 6.4).

PIMO is similar to SKOS or Topic Maps (TM) [1] in its goal of providing an easy way of modelling, but different in the way concepts are modeled. RDFS classes and sub-class relations (which are used in NRL) are used to represent the classes of Things, individuals are represented using typed RDF resources. From TM, the idea of indirect identification is taken up.

PIMO is different from OWL, where `sameAs` relations are symmetric equivalence relations forming a fully connected graph amongst identical resources. In PIMO, Things are connected to their equivalent resources using directed relations.

The design rationale is to keep the PIMO ontology as minimal as possible, and also the data needed to create a PIMO for a user as minimal as possible. Inside one PIMO of a user, duplication is avoided. PIMO builds on Semantic Desktop ontologies (NRL, NIE, NAO) and guidelines are provided how to reuse other existing RDFS ontologies (Section 9) and import data expressed in these ontologies. After importing, rules are used to integrate facts (see Section 7.2).

Only by addressing all key issues — precise representation, easy adoption, easy to understand by users, extensibility, interoperability, reuse of existing ontologies, data integration — PIMO provides a framework for creating personal information management applications and ontologies.

3.1 Downloading PIMO

The RDF descriptions of the ontology can be retrieved from its namespace using content negotiation:

```
Namespace:
http://www.semanticdesktop.org/ontologies/2007/11/01/pimo#
```

Different serializations are available:

- XML/RDFS Serialization: PIMO (Data Graph Only)
  `http://www.semanticdesktop.org/ontologies/2007/11/01/pimo/pimo_data.rdfs`
- XML/RDFS Serialization: PIMO (Metadata Graph Only)
  `http://www.semanticdesktop.org/ontologies/2007/11/01/pimo/pimo_metadata.rdfs`
- TriG Serialization: PIMO (Graph Set)
  `http://www.semanticdesktop.org/ontologies/2007/11/01/pimo/pimo.trig`

4 PIMO integrates with key ontologies

The PIMO framework does not stand alone but is part of a set of carefully designed and integrated ontologies for the Semantic Desktop which were developed in parallel to provide an integrated framework. In Figure 1 an overview is given.

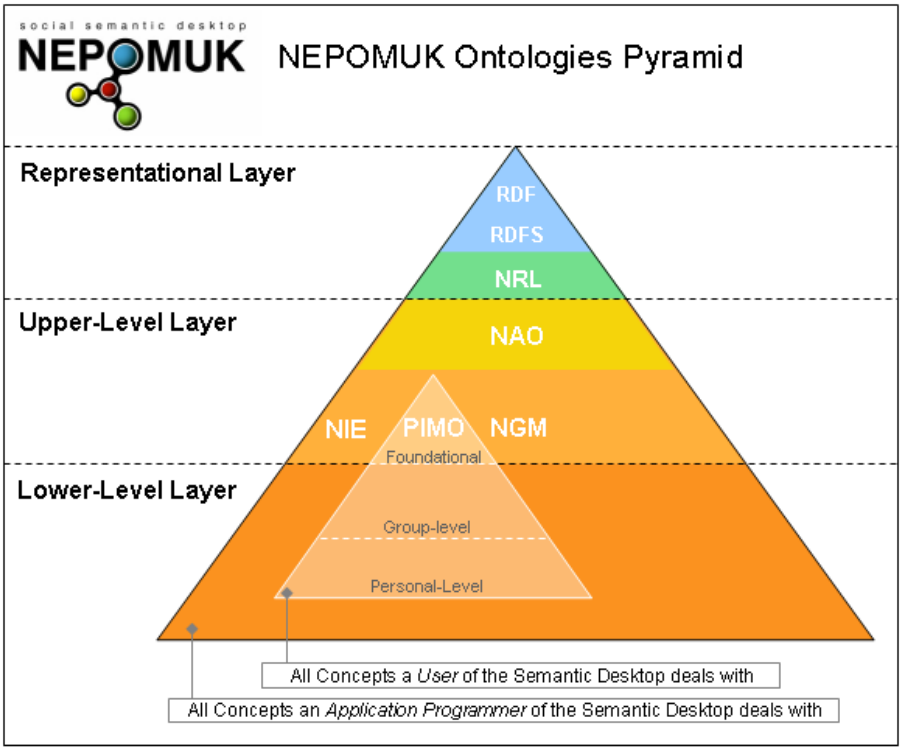


Figure 1: Integrated Ontologies

The NEPOMUK Representational Language **NRL ontology**[6] builds the representational layer and the semantic metadata axioms used to express PIMO. NRL is a meta-language comparable to OWL or RDF/S. The key characteristics of NRL are (on top of RDF/S) support for named graphs in semantic statements, a notation for contextualized inference and semantics, and a selection of semantic relations (inverse, transitive, reflexive).

The NEPOMUK Information Element **NIE ontology**[7] describes desktop elements such as address book entries, documents, e-mails, appointments, pictures, and multimedia files. The ontology discriminates between the representation (binary files) and the information stored therein. PIMO reuses

[6] `http://www.semanticdesktop.org/ontologies/nrl/`
[7] `http://www.semanticdesktop.org/ontologies/nie/`

the classes of NIE and suggests how to integrate and annotate vast personal spaces of information (PSI) expressed in NIE.

Annotations and tagging are represented in the NEPOMUK Annotation **NAO ontology**. It represents change-dates, author, and other key metadata of documents. NIE and PIMO both extend NAO. A part of NAO is the NEPOMUK Graph Metadata **NGM ontology** to annotate named graphs.

The PIMO ontology crosses two of the layers in the ontology pyramid, data related to PIMO can be divided into three smaller layers (also visible in Figure1):

- *Foundational PIMO*: The PIMO ontology as such, as defined in this specification and accompanying NRL serialization. This includes the classes and properties of *PIMO-upper* (see Section 6.12), which work on the *Upper-Level Layer* of NEPOMUK and everything else defined in the PIMO vocabulary. PIMO-upper is the same for every Semantic Desktop user and is valid in a global context.
- *Group-level PIMO*: Domain ontologies that are shared within a group. They can be imported by users. A description is given in Section 8.
- *Personal-level PIMO*: The classes, properties, and Things created by an individual user. Also called *user-PIMO*, this layer includes the data that is only relevant in the context of one individual.

5 Examples

A scenario is used to explain the ontology elements. A fictional persona, Claudia Stern, is our example user. She is working for EX-Ample Inc., a fictional company producing "**Ex**treme Guitar **Ampl**ifi**e**rs", and her current task is to organize a business trip to a meeting with guitarists and bass players in Belfast.

For convenience and readability, this specification uses an abbreviated form to represent URI-References. A name of the form `prefix:suffix` should be interpreted as a URI-Reference consisting of the URI-Reference associated with the prefix concatenated with the suffix.

RDF graphs are written in N3/Turtle syntax. Examples serialized as RDF appear in this typesetting:

```
claudia:Claudia a pimo:Person;
 pimo:isDefinedBy claudia:PIMO;
 nco:hasEmailAddress <mailto:claudia@example.com> .
```

5.1 PIMO ontology and namespaces

The ontology described in this document has this namespace:

```
Namespace:
http://www.semanticdesktop.org/ontologies/2007/11/01/pimo#
```

During the lifetime of the NEPOMUK project (until Dec 2008), the PIMO ontology and the according documentation may change, but the namespace will not change. The namespace stays fixed to keep the necessary changes of software implementations at a minimal level. We have adopted this practice from other projects, which have applied it successfully. Examples are the W3C's XSD datatypes (the recommendation changed in 2007, the namespace did not) or the FOAF project.

Throughout this document these ontologies and namespaces are used, also indicating their respective versions PIMO is building on:

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix nrl:  <http://www.semanticdesktop.org/ontologies/2007/08/15/nrl#>.
@prefix nao:  <http://www.semanticdesktop.org/ontologies/2007/08/15/nao#>.
@prefix pimo: <http://www.semanticdesktop.org/ontologies/2007/11/01/pimo#>.
@prefix ncal: <http://ont.semanticdesktop.org/ontologies/2007/04/02/ncal#>.
@prefix nco:  <http://ont.semanticdesktop.org/ontologies/2007/03/22/nco#>.
@prefix nfo:  <http://ont.semanticdesktop.org/ontologies/2007/03/22/nfo#>.
@prefix nie:  <http://www.semanticdesktop.org/ontologies/2007/01/19/nie#>.
@prefix nmo:  <http://www.semanticdesktop.org/ontologies/2007/03/22/nmo#>.

@prefix claudia: <http://www.example.com/people/claudia#> .
```

6 Creating Personal Information Models

In this section, all key elements of the ontology are presented. The order used reflects the steps a knowledge engineer will have to take to implement this recommendation.

6.1 The User and their Individual PIMO

As a prerequisite to create a PIMO and Things inside the PIMO, each user needs a *personal namespace*. The namespace is used as a prefix for all URIs minted for the user. Often these are namespaces using the HTTP URI scheme, but any RDF namespace can be used. The example namespace used in this document is `http://www.example.com/people/claudia#` and is abbreviated with "`claudia:`".

Users are represented as instances of the class `pimo:Person`. For each instance, a new URI is generated and a few key facts are represented to identify the user. After the user has been instantiated, details such as email addresses are added by using terms from the NEPOMUK contact ontology, NCO. In NCO, contact information connected to people is modeled as a complex resource, not as a simple literal:

```
claudia:Claudia a pimo:Person;
 rdfs:label "Claudia Stern";
 nco:hasEmailAddress mailto:claudia@example.com.

mailto:claudia@example.com a nco:EmailAddress;
 nco:contactMediumComment "work";
 nco:emailAddress "claudia@example.com".
```

The second entity that needs to be represented is the *Personal Information Model of the User*. It is connected to the user via the `pimo:creator` relation, and the user's namespace is added. For Claudia this is:

```
claudia:PIMO a pimo:PersonalInformationModel;
  pimo:creator claudia:Claudia;
  nao:hasDefaultNamespace "http://www.example.com/people/claudia#";
  nao:hasDefaultNamespaceAbbreviation "claudia".
```

`pimo:PersonalInformationModel` is a sub-class of `nrl:Ontology`, allowing more metadata to be added using NRL compliant standards. More about NRL metadata is described in Section 11.9. We further call a this instance of `pimo:PersonalInformationModel` of an individual a *user-PIMO*. Claudia's user-PIMO is `claudia:PIMO`. As an abbreviation, it is also correct to write "Claudia's PIMO" instead of "Claudia's user-PIMO".

6.2 Things

The PIMO ontology defines the basic class `Thing` for mental concepts. Every information element encountered in knowledge work by a user is represented as a Thing. A Thing is a unique representation of an entity of the real world within one user-PIMO. On the PSI of a user, a real world entity can be represented in multiple data sources. For example, the person "Dirk Hagemann" may be author of an e-mail, described in an address book entry, and stored in a accounting tool, all part of the workspace of "Claudia". One instance of `pimo:Person` is created as unique `Thing` linking to these multiple representations, such as shown in Figure 2.

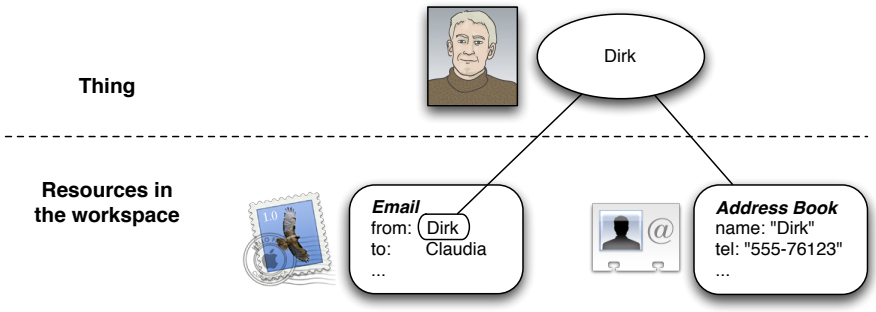


Figure 2: Thing and Resources

An application handling a resource in the workspace can be aware that there may be a Thing representing the resource. For example, Claudia's e-mail client may examine the sender of an e-mail (Dirk) and search for the `pimo:Thing` that represents Dirk uniquely. Once the right Thing is found by the application, more information about Dirk can be discovered.

This principle includes all elements (`nie:InformationElement` and other RDF resources) in the user's *Personal Space of Information (PSI)*. For each information element, a Thing in the user's PIMO must be created. The information element exists independent of the user, the same element can be stored in multiple folders or data sources on one desktop, and also on other desktops and on the web. A Thing is the personalized view of *one user* on this information element, independent of representation or storage location.

To be adequate, a PIMO of a user should contain all nameable entities known to the user, but to be efficient, this representation should be restricted to the minimal data needed.

6.3 Connecting Things to the User's PIMO

In a scenario of multiple connected semantic desktops, it will frequently occur that users import data from each other's desktop onto their own desktop. It is therefore important to know which resources (primarily Things, but also Classes and Properties) were created by which user and originate from which PIMO. For this, the property `pimo:isDefinedBy` is used.

Continuing the example above, this property connects the Person to the PIMO in which it is defined. This is mandatory for every defined Thing and allows applications to identify which elements are part of a user-PIMO and which are not[8].

[8]We intentionally did not only rely on NRL graphs to model the relation between the model and instances defined by it. A graph can only contain *statements*, about, but not *resources* as such. To model that a resource is part of a PIMO, the `pimo:isDefinedBy` relation is a clear representation. Additionally, named graphs can be used to declare what *statements* are declared in a PIMO,

```
claudia:Claudia pimo:isDefinedBy claudia:PIMO.
```

An `isDefinedBy` property is also defined in RDFS, where resources can be connected to their defining ontologies, and is also discussed in the light of the OWL standard[9]. The semantics of `isDefinedBy` in PIMO is based on these, with the extension that we define it as a required property.

6.4 Identification of Things

A Thing *must* have an URI and *should* be described with properties that identify it. Identifiers allow to analyse information elements and find occurrences of the Thing. For example, the person "Dirk Hagemann" is represented once as an instance of the class `pimo:Person` and identified using his e-mail address. The RDF descriptions of emails and documents can then be analysed to find resources that represent the same entity via this identifier.

```
# The canonical Dirk
claudia:DirkHagemann a pimo:Person;
 pimo:isDefinedBy claudia:PIMO;
 nco:hasEmailAddress <mailto:dirk@example.com>.

# An e-mail in which Dirk #2 occurs
<imap://claudia@example.com/INBOX/1> a nmo:Mail;
 nmo:from <imap://claudia@example.com/INBOX/1#from>.

# Dirk #2, the email sender
<imap://claudia@example.com/INBOX/1#from> a nco:Contact;
 nco:hasEmailAddress <mailto:dirk@example.com>.
<mailto:dirk@example.com> a nmo:EmailAddress;
 nco:emailAddress "dirk@example.com".

# Dirk #3, as address book contact
<file://home/claudia/dirk.vcf#dirk> a nco:PersonContact;
 nco:nameFamily "Hagemann";
 nco:nameGiven "Dirk";
 nco:hasEmailAddress <mailto:dirk@example.com>;
 nco:photo <http://www.example.com/people/dirk/photo.jpg>.
```

In this example, we see that the Person Dirk appears three times in this knowledge workspace. First, in the form of an instance of `pimo:Person`, as the canonical Dirk. Second, as sender of an e-mail and third as entry in an address book. Only one instance is the `pimo:Thing` representation of Dirk: `claudia:DirkHagemann`. The others are information elements representing the same entity.

To work effectively, PIMO is based on the *Unique Name Assumption* (UNA). The UNA is a rule that declares two RDF resources with different URIs as different individuals. This is common in desktop applications (for example files with different names are different) and intuitive to grasp for users. But it is different from the OWL ontology language where duplicate entries are common and the UNA is not enforced. PIMO is designed for personal systems, where an application has access to the complete model and can avoid duplicates before creating them.

To enforce the UNA, duplicate Things *must* be avoided. The crucial moment to do this is before creating a new Thing. Things can either be created by the user manually or automatically by analysing existing native resources. In any case, before creating a new Thing, all existing Things have to be examined if a

see 11.9

[9] `http://www.w3.org/2007/OWL/wiki/Syntax#Declarations_and_Structural_Consistency`

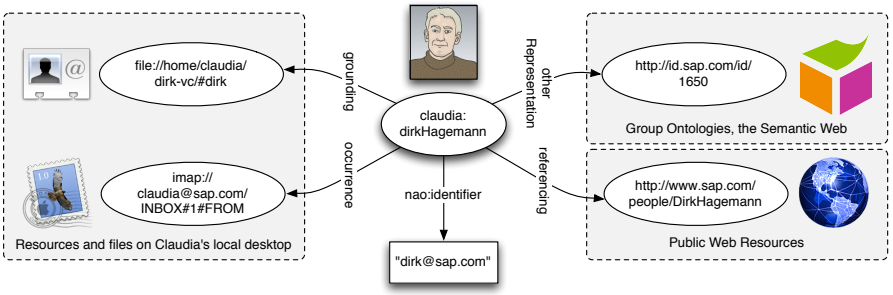


Figure 3: Different Identification Mechanisms

Thing with a same name or same identifying properties already exists. If an existing Thing is found, it *must* be reused. Immediately after creating a new Thing, identifying properties should be set to distinguish the Thing and avoid duplication. Section 11.1 further describes the complete process of creating things.

In the next paragraphs, essential identifying properties are described, an overview is given in Figure 3.

The primary identifier of a Thing is it's URI. New URIs for Things *must* be generated using the namespace of the user as prefix and then a unique local name. Although the local name can be entirely a random string, we recommend to include the label in the URI for readability. When minting a new URI that clashes with an existing URI, a random element can be added to the new URI. A URI for Claudia Stern is:

```
claudia:Claudia
```

NAO-Identifiers Existing identification schemes based on NAO should be reused for this purpose by representing them with `nao:identifier` and its sub-properties. If an identifier is found as meta-data of a native resource (usually an `nie:InformationElement`), the identifier *must* be copied to the Thing. This allows others to match and identify the correct Thing when encountering the next information element. Example identifiers are `nmo:messageId` for e-mails, `ncal:uid` for appointments, or `nexif:imageUniqueID` for images. Instead of using the plain `nao:identifier` property, these specific properties should be used or new sub-properties of `nao:identifier` created [10]. In this document, we assume that e-mail addresses can be used to identify persons.

```
# Copy all identifiers you can find about the Thing.
claudia:DirkHagemann a pimo:Person;
 nao:identifier "dirk@example.com".
```

Identifiers consisting of multiple RDF statements cannot be captured using `nao:identifier`. They are comparable to a primary key in a relational database consisting of multiple columns. These **multi-key identifiers** *must* be merged into one `nao:identifier`.

Grounding Occurrence The relation `pimo:groundingOccurrence` is used to link a Thing to an `nie:InformationElement` that has this Thing as primary topic. For example, the grounding for a person could be the entry in the address book describing the person. On the other hand, an e-mail with this person as the sender or recipient would normally not be a grounding occurrence. A Thing represents the mental concept, the `pimo:groundingOccurrence` links to

[10] I.e. if you want to represent ISBN numbers and there is no property for them, create a new sub-property `isbn` of `nao:identifier`.

existing Information Elements that are handled by existing applications. This is a key for reusing the features of these applications. The grounding occurrence can change, for instance if a file was moved and the URI of the Information Element changed, the grounding occurrence relation needs to be changed. A similar case happens when a file is uploaded to a shared workspace and not kept locally any more — all annotations of the Thing stay the same (the URI of the Thing does not change), the Information Element changes. Multiple values are allowed, this reflects the fact that the same Thing can be represented in multiple applications, and dependent on the work context, the user may want to open a different application.

```
# Link to Dirk #3 from example above.
claudia:DirkHagemann a pimo:Person;
 pimo:groundingOccurrence <file://home/claudia/dirk.vcf#dirk>.
```

Occurrence The relation `pimo:occurrence` connects a `pimo:Thing` with a resource representing the same real world entity. Facts about the occurrence are then also valid for the connected Thing. For example, if the person Dirk appears as sender of an e-mail, then the resource identifying the sender is an *occurrence* of Dirk. Based on the occurrence relation, Dirk (the unique `pimo:Person`) is the sender of the given e-mail. Occurrence relations are to be used on resources *representing* the same entity in a different context, but not on resources *mentioning* the entity. For example, it is not valid to say that an e-mail is an occurrence of a person, only the sender or recipient can be occurrences of a person.

Occurrences of a Thing can be found by searching for entities with the same identifying properties.

```
# Link to Dirk #2 from example above,
# he occurs as sender of an e-mail
claudia:DirkHagemann a pimo:Person;
 pimo:occurrence <imap://claudia@example.com/INBOX/1#from>.
```

Besides identification, both the `pimo:groundingOccurrence` and the `pimo:occurrence` relation have implications on data integration and affect semantic meaning of a Thing. This will be described later in Section 7.

Referencing Occurrence A Referencing Occurrence is an indirect approach to identification. Annotating a Thing with an information element as *referencing occurrence* states that the information element contains a description of the Thing. Its primary topic must be the Thing. The Thing is indirectly identified by the element, when two Things in different models share the same information element as referencing occurrence, they may be equal and could be matched. The following description is an adaption of XTM's subject indicators [18, 15]. The referencing occurrence is a kind of proxy for the Thing. Examples of referencing occurrences are:

```
claudia:DirkHagemann a pimo:Person;
 pimo:referencingOccurrence <http://www.example.com/people/DirkHagemann>.

claudia:ExampleInc a pimo:Organization;
 pimo:referencingOccurrence <http://www.example.com/>;
 pimo:referencingOccurrence <http://en.wikipedia.org/wiki/Example.com>.
```

It should contain a human readable documentation describing the concept. The resource could be a document, ontology, video, audio, anything able to describe to a human what the concept is about. The resource is a reference to the concept of the Thing. A good example for a referencing occurrence is a wikipedia article.

A referencing occurrence describes the concept with the purpose of being widely used by ontologies. Consequently, it is important that the document describes exactly what concept it is about and what not. Even if the author works as accurately as possible, different people will never interpret a referencing occurrence 100% the same way. However, the concept of referencing occurrences is worth using it, because it allows a shallow match of heterogenous information models, and because there is finally no alternative to it.

It is recommended to use wikipedia URIs as objects of referencing occurrences. In contrast, URIs minted by *DBPedia*[11] *must* be related using the `pimo:hasOtherRepresentation` relation.

Other Representation The `pimo:hasOtherRepresentation` relation is used to connect `pimo:Things` with other representations of the same Thing in other Semantic Web ontologies. This can be the case with shared ontologies, such as company white page systems or Semantic Social Networking websites.

The knowledge modeled should be compatible with the ontologies used by the user. An example for such other representation is[12]:

```
claudia:DirkHagemann a pimo:Person;
 pimo:hasOtherRepresentation <http://id.example.com/person/1650>.
```

Another example would be the city of Belfast where Claudia wants to travel to, linked to the DBPedia entry about it:

```
claudia:Belfast a pimo:City;
  pimo:isDefinedBy claudia:PIMO;
  nao:prefLabel "Belfast";
  nao:personalIdentifier "Belfast";
  pimo:hasOtherRepresentation <http://dbpedia.org/resource/Belfast>;
  geo:lat "54.5833333";
  geo:long "-5.9333333".
```

The relation can be used both to identify Things by their other representations, and to fetch more data. In this example, the latitude and longitude are actually superfluous data, as they can be retrieved from the other representation in DBPedia. Assuming Dirk also represents Belfast in his PIMO, but independent from Claudia, but linking to the same DBPedia entry, algorithmically matching their different representations is straightforward.

Other Conceptualization To map user-generated classes to classes defined in other ontologies, the `pimo:hasOtherConceptualization` relation connects classes defined in a user's PIMO with classes defined in domain ontologies. Classes defined in domain ontologies should be sub-classes of PIMO-upper classes, see Section 9.

Implementations can use the `pimo:hasOtherConceptualization` to allow the user and algorithms to map user–specific classes to classes defined in other ontologies, without implying that there is a sub-class relationship.

6.5 A Complete Example

A complete example for all different identification properties can now be built from the above annotations.

[11] DBPedia is a Semantic Web representation of wikipedia and provides URIs for concepts, whereas wikipedia provides URIs for documents describing concepts. An example DBPedia URI is: `http://dbpedia.org/page/Berlin`

[12] Using the URI scheme of the ECS University in our example domain. `http://id.ecs.soton.ac.uk/docs/`

For Claudia, her co-worker Dirk Hagemann is identified and linked to occurrences like this:

```
# The canonical pimo:Person Dirk,
# a pimo:Thing from Claudia's PIMO
claudia:DirkHagemann a pimo:Person;
 pimo:isDefinedBy claudia:PIMO;
 nao:prefLabel 'Dirk Hagemann';
 nao:identifier "dirk@example.com";
 pimo:occurrence <imap://claudia@example.com/INBOX/1#from>;
 pimo:groundingOccurrence <file://home/claudia/dirk.vcf#dirk>;
 pimo:referencingOccurrence <http://www.example.com/people/DirkHagemann>;
 pimo:hasOtherRepresentation <http://id.example.com/person/1650>.


# An e-mail in which Dirk #2 occurs
<imap://claudia@example.com/INBOX/1> a nmo:Mail;
 nmo:from <imap://claudia@example.com/INBOX/1#from>.

# Dirk #2, as email sender
<imap://claudia@example.com/INBOX/1#from> a nco:Contact;
 nco:hasEmailAddress <mailto:dirk@example.com>.

<mailto:dirk@example.com> a nmo:EmailAddress;
 nco:emailAddress "dirk@example.com".

# Dirk #3, as address book contact
<file://home/claudia/dirk.vcf#dirk> a nco:PersonContact;
 nco:nameFamily "Hagemann";
 nco:nameGiven "Dirk";
 nco:hasEmailAddress <mailto:dirk@example.com>;
 nco:photo <http://www.example.com/people/dirk/photo.jpg>.
```

This allows implementations to:

- identify the Thing when found occurring in documents,
- open a grounding occurrence to see the Thing within an existing desktop application (i.e. the address book entry for a person),
- match this Thing with other representations via the same referencing occurrence,
- use the other representation from the company's white pages to show additional data about the Thing.

The `pimo:occurrence` link is the generic basis, `pimo:groundingOccurrence` and `pimo:hasOtherRepresentation` are sub-properties of it. This data should be generated automatically and unsupervised. Adding identifying properties to a Thing helps to find more occurrences and therefore more information about it.

6.6 Labels and Names of Things

To label Things, we recommend the NEPOMUK Annotation Ontology (NAO) vocabulary. It defines properties for a *preferred label*, *multiple alternative labels*, and a *personal identifier*.

`nao:prefLabel` *A preferred label for a Thing.* This property **must** be applied to every instance of `pimo:Thing`. It can be used by applications to represent the Thing with a textual label and should be human-readable. There must

only be one `prefLabel` per Thing (mincardinality and maxcardinality should be one)[13].

`nao:personalIdentifier` *Defines a unique personal label for a Thing.* The label *must* be unique within the scope of a user. If both are used, `nao:personalIdentifier` and `nao:prefLabel` of a resource *should* have the same value. Note that it is not a sub-property of `nao:prefLabel`, both have to be set explicitly.

Personal identifiers are the *recommended* way to label and identify Things. As they are unique and human-readable, they *may* be used for multiple application scenarios such as wikis, tagging, or matching terms found in free-text.

`nao:altLabel` *An alternative label alongside the preferred label for a Thing.* These are alternative spellings, translations, nick-names. Implementations can use these labels to find Things when the user enters a text in a search box or when analysing free text. If a Thing has occurrences, the labels of occurrences *should* be copied as alternative labels to the thing.

In combination, these labelling techniques allow applications to clearly label Things in user interfaces but also to lookup for Things based on alternative names. For our example, these are:

```
claudia:DirkHagemann a pimo:Person;
# preferred label when shown
 nao:prefLabel 'Dirk Hagemann';
# a nickname for Dirk
 nao:altLabel "hacki";
# a common misspelling
 nao:altLabel "Dirck Hagemann";
# the personal identifier
 nao:personalIdentifier "DirkHagemann".
```

Additionally, visual cues (icons, images, thumbnails) can be attached by using NAO symbol relations:

- `nao:hasSymbol`
- `nao:prefSymbol`
- `nao:altSymbol`

6.7 Textual description of Things

To describe Things with a free-text, the simple `nao:description` property should be used. This allows users to add a (possibly searchable) description of the Thing in a simple way. The description string value should contain no format markup but be a plain text.

For more complex free-text descriptions of Things, the property `pimo:wikiText` is reserved. Formatting (font-weight, italics) and linking to other pages is supported in this property, implementers may use the *Wiki Interchange Format* [14] as syntax.

[13] These restrictions are not explicitly noted in the RDF description of the property, as NRL does not support property restrictions for classes.

[14] `http://semanticweb.org/wiki/Wiki_Interchange_Format`

6.8 Rating and Ranking Things

Ratings of Things can be expressed using `nao:numericRating`. For numericRating, the range of values *must* be within $[0..1]$ (inclusive). Applications *may* partition the values into discrete ratings (such as 0.2, 0.4, 0.6, 0.8, 1.0 to represent the semantics of "5 star ratings").

The rating values may and should be used for *ranking* of Things and filtering. A populated PIMO contains thousands of Things, user interfaces should use the `nao:numericRating` values to filter out low-ranked resources and highlight high-ranked resources. Implementations *can* set the `nao:numericRating` values automatically to computed values.

6.9 Modelling Time

In PIMO, no special treatment of time is modeled. We are aware that representing points in time, durations, and other periods of time is an important aspect of ontologies. We recommend to use the XML Schema Datatypes to represent time. There, ISO 8601 is recommended. Timezones must be handled according to this standard, encoded inside the literal value[15].

Periods of time can be represented using sub-classes of the abstract class `pimo:ProcessConcept` which represents lasting processes such as events or projects. For durations that last a number of days or months, we recommend to use the standardized XML datatypes[16]:

- `xs:dayTimeDuration` for durations measured in days, hours, and minutes.
- `xs:yearMonthDuration` for durations measured in months and years

Implementers are free to use either the XSD types or sub-classes of `pimo:ProcessConcept`.

There have been issues with other notations of duration and therefore the W3C Semantic Web Best Practices and Deployment Group published a note[17] to restrict durations to these values.

6.10 Representing Modification and Change Dates

The change and creation dates of Things are important metadata for personal information management applications. Knowing about recent changes is an important cue for users to retrieve documents. Many applications offer the feature to show recent changes or filter by them. Consequently, it has to be straightforward, simple, and fast to query for the modification dates.

The NAO properties `nao:created`, `nao:modified`, and `nao:lastModified` *shall* be used to track the change dates of Things. Creation and modification allow only one, modification allows multiple date values. Created and lastModified values *must* be set for each Thing, at least one modified value *must* be set. These values are intended for resources of type `pimo:Thing`, `pimo:Association`, `rdfs:Class`, and `rdf:Property` when created by the user.

[15] For a detailed representation of time events, refer to the NIE documentation, where timezones are discussed (`http://www.semanticdesktop.org/ontologies/2007/04/02/ncal/#sec-tzd`). NIE represents time using the NcalDateTime class and its properties date, dateTime, ncalTimezone. Timezones are represented using a Timezone class, that is inspired by RFC 2445.

[16] The XS namespace is `http://www.w3.org/2001/XMLSchema`, but the two duration datatypes are defined in the XPath recommendation in 2007, see `http://www.w3.org/TR/xpath-functions/#dt-dayTimeDuration`

[17] `http://www.w3.org/TR/swbp-xsch-datatypes/#section-duration` Since XPath 2.0 has become a W3C recommendation in January 2007, this note is partly obsoleted.

Example:

```
# Represent modification dates of a Thing
claudia:DirkHagemann
  nao:created "2007-10-26T15:23:01";
  nao:modified "2007-10-26T15:23:01";
  nao:modified "2007-10-29T08:04:30";
  nao:lastModified "2007-10-29T08:04:30".
```

The **semantics** of these dates is that the description of the Thing has changed, facts about the Thing have been added, removed, or modified. Changes to `pimo:objectProperty` (`pimo:related`, `pimo:hasTopic`, etc.) or `pimo:datatypeProperty` (`name`, `address`, `label`, etc.) imply such a modification. Included are also changes to the labels (`nao:prefLabel`, `nao:altLabel`, `nao:personalIdentifier`). Modification of any other statement (such as `pimo:definedBy`, `nao:modified`) do not imply a modification nor a change of dates. As current RDF stores a priori do not support automatic tracking of changes, applications have to implement housekeeping of these dates, or use services for tracking. [18]

6.11 Setting the Class of a Thing

All Things are of type `pimo:Thing` or one of its sub-classes. The PIMO ontology itself defines several sub-classes such as `pimo:Person` or `pimo:Organization`. If these are not specific enough, the user can either create new sub-classes manually (see Section 6.17), or import group-level ontologies (Section 9).

As a rule of thumb, the question to be answered by assigning the class is "*What is this Thing?*". In comparison to OWL, where classes are commonly based on the properties of the object ("vegetarians are entities not eating flesh") , classes represent the type (the *nature*) alone.

It is also recommended to only use one explicit class for a Thing. The wish to add multiple classes is often an indication that some classes can be better modeled using relations. For example, it is recommended to use a datatype property *"is this person a vegetarian? yes or no"* and explicitly set it instead of assigning a sub-class of Person called *Vegetarian*. If Things with two classes are needed (for example, if something is both a "Car" and a "Locatable") then the preferred way is to change the class model (make Car a subclass of Locatable or create a new class "LocatableCar" with both as superclass) than to add both types to one Thing. Nevertheless, it is not forbidden to add two types.

When a Thing has occurrences that are expressed in the NEPOMUK Information Element ontology (NIE), suitable mappings from NIE classes to PIMO classes are available in a mapping file [19].

6.12 The PIMO-upper ontology

PIMO contains an *upper ontology* for basic concepts in Personal Information Management (PIM): Person, Location, Event, Organization, Topic, Document, Time. They are modeled to answer basic questions about a Thing:

- Who is associated? Person

[18] The value of lastModified is redundant as it could be computed by sorting the modified dates during query time, but this is not possible without nested SPARQL queries, a $max()$ function, and grouping, none of these part of the SPARQL standard. The stores that support such non-standard operations still need time to compute the value. As the last modification date is very important for applications to assist users finding information, the redundancy is intended.

[19] `http://www.semanticdesktop.org/ontologies/2007/11/01/nietopimomapping.rdf`

- Where is this? Location
- When is it? Time
- What is it about? Topic

The classes are the *foundational part* of PIMO in the *Upper-Level Layer* of the overall NEPOMUK ontologies as shown in 1. This level serves as integration point for PIM applications, in the broader perspective of the Semantic Desktop, the classes can serve as upper classes for group– and domain–level ontologies (see Sect. 9).

6.13 Classes in PIMO-Upper

The classes have been defined based on related ontologies, a user study, and several software prototypes that have been evaluated. Figure 4 gives a rough overview of the available classes.

Thing The root class of the upper ontology. Every entity that can be in the direct attention of the user is a Thing.

Collection A collection of Things, independent of their class. The items in the collection share a common property. Several usability studies showed that collections are important for PIM.

Group A group of Persons. They are connected to each other by sharing a common attribute, for example they all belong to the same organization or have a common interest.

Location A physical location. Sub-classes are modeled for the most common locations humans work in: Building, City, Country, Room, State. This selection is intended to be applicable cross-cultural and cross-domain. City is a prototype that can be further refined for villages, etc.

LogicalMediaType MediaConcepts are logical media types (e.g., a book, a contract, a promotional video, a todo list). The user can create new logical media types dependent on their domain: a salesman will need MarketingFlyer, Offer, Invoice while a student might create Report, Thesis and Homework.

Organization An administrative and functional structure (such as a business or a political party).

Person Represents a person. Either living, dead, real or imaginary. In this regards, similar to `foaf:Person`.

ProcessConcept Concepts that relate to a series of actions or operations conducing to an end. Sub-classes are defined for Event, SocialEvent, Meeting, Project, and Task.

Topic A topic is the subject of a discussion or document. Topics are distinguished from Things in their taxonomic nature, examples are scientific areas.

These classes are intentionally kept generic. More specialized ontologies should be used for certain domains of application, see Section 9. The classes of these ontologies are then sub-classes of upper ontology classes.

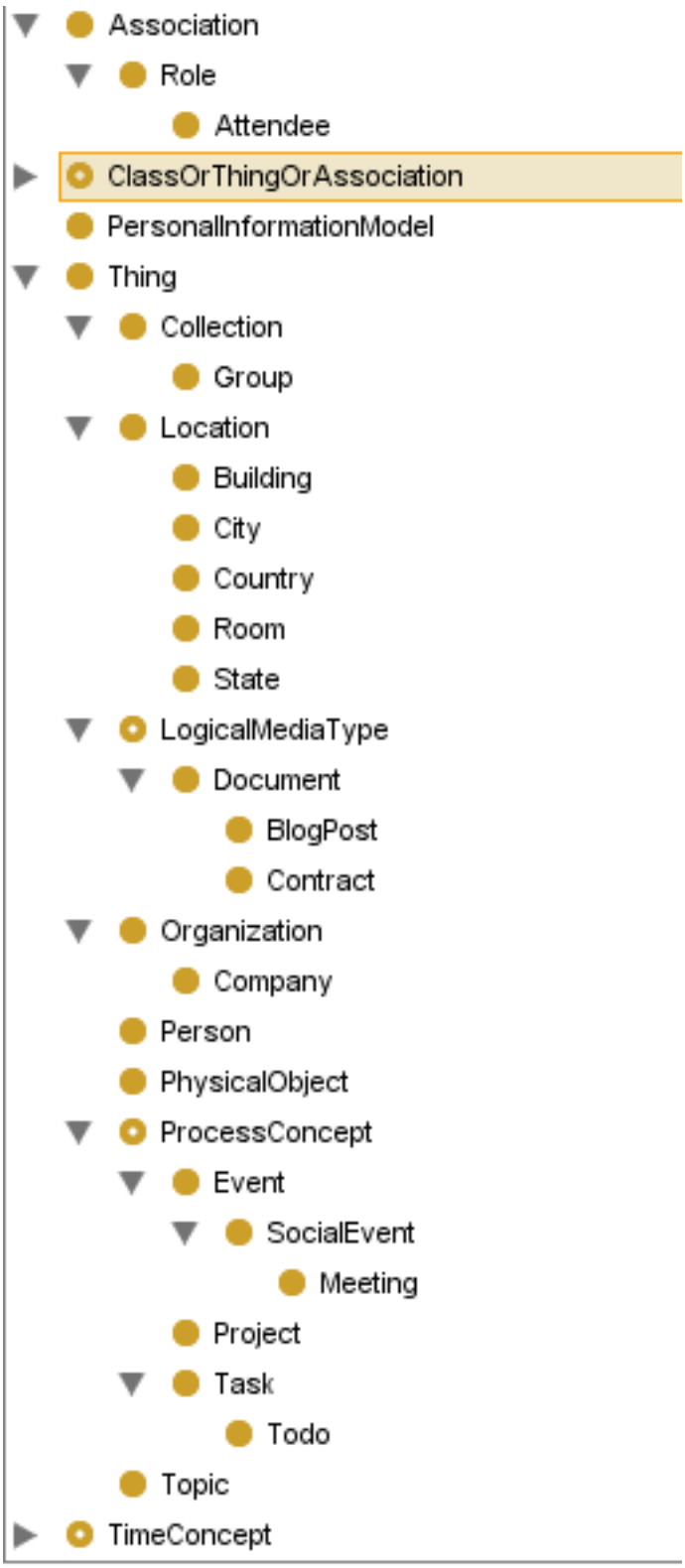


Figure 4: Classes in PIMO-Upper

6.14 Describing Things with Attributes and Relations

Conventional RDF statements are used to describe Things. Predicates have to be defined as `rdfs:Properties` according to the NRL standard. Alternatively, it is also possible to use properties from other modeling languages like OWL or RDFS although we do not encourage this without a proper mapping of existing ontologies to PIMO (see 9).

Properties that are intended to be editable and visible to end users *must* be sub-properties of either `pimo:datatypeProperty` or `pimo:objectProperty`.

Many NIE and NAO properties can be used from PIMO Things, see Section 11.10.

6.15 Generic Properties in PIMO-Upper

The PIMO-upper ontology contains basic relations between Things and a few core attributes for identifying them (described above in Sect. 6.4). These sub-properties of `pimo:objectProperty` are:

`pimo:related` is the most generic relation, giving no further indication how Things may be related. Related is defined to have itself as inverse property, it is indirectly a `nrl:SymmetricProperty`, but does not inherit this attribute to sub-properties. Sub-properties can be asymmetric, depending on the inverse-of relation they define.

`pimo:hasPart` and `pimo:partOf` model partitive relations. They are inverse. Neither is transitive, because part-of relations used for modelling in the domain of Personal Information Management are vague due to the many contexts of interpretation (a hotel may be part of a trip plan, a trip plan part of a project, but this does not indicate the hotel to be part of the project).

`pimo:hasTopic` and `pimo:isTopic` connect a Thing of interest with a Thing reflecting about it. For example, a meeting can have a project as a topic, or a meeting has a document as a topic, when the goal of the meeting is to discuss the document. After the meeting, the meeting minutes are a new Thing having the meeting as a topic. This is not restricted to meetings but also an organization or a person can have a certain technology as a topic to express that they are working on the topic. The relation is not transitive, not symmetric. It is not asymmetric because a document A may have document B as topic, and B also A.

Implementers may use these generic properties directly, or create sub-properties of them or sub-properties of the more generic `pimo:objectProperty`. The main reason to have the generic properties is the semantic meaning of the relations, which can help to create user interfaces or model domains. Ontology authors can ask themselves “does a new property model a part-of relation or not, does it assign a thing with a topic, or is it a generic relation?” and then extend one of the generic properties.

For these generic relations, specialized sub-properties are defined when used on specific classes in the PIMO upper ontology.

6.16 Refined properties in PIMO-Upper

Additional to the above relations, semantically interesting relations between PIMO upper classes are modeled. Especially those which can be used as symmetric or transitive relations for inference.

`pimo:narrower` and `pimo:broader` relate Topics to each other. As Topics are an important mean to organize document collections based on a taxonomy, these two predicates are defined. They are inverse of each other and transitive.

`pimo:hasOrganizationMember` and `pimo:isOrganizationMemberOf` are relations connecting a Person to an Organization.

`pimo:hasLocation` and `pimo:isLocationOf` relate a locatable Thing with its Location. Locatable is an abstract sub-class of Thing.

`pimo:containsLocation` and `pimo:locatedWithin` relate two locations within each other. Note that for geographic locations representing a physical space, inclusion is transitive.

6.17 Creating Personalized Classes and Properties

The predefined classes and properties are intended as a generic basis to be extended. The user can always create new classes and property types, or existing ontologies can be imported (see Section 8). A number of requirements apply:

- The superclass has to be `pimo:Thing` or a sub-class.
- The class has to be labelled with `nao:prefLabel`.
- The class has to be related to the user's PIMO with `pimo:isDefinedBy`.

Similarly for custom properties:

- The property has to be labelled with `nao:prefLabel`.
- The property has to be related to the user's PIMO with `pimo:isDefinedBy`.

For properties that relate two things, the following applies:

- The property *must* be a sub-property of `pimo:objectProperty` (either directly or indirectly via inference).
- The super-property *should* be one of `pimo:related`, `pimo:hasTopic`, `pimo:isTopicOf`, `pimo:hasPart`, or `pimo:partOf`.
- An inverse property *must* be defined. Inverse properties define the semantic meaning in both ways, which is required for user interfaces showing relations.

For custom properties that have *a literal or datatype as range* the following applies:

- They must be a sub-property of `pimo:datatypeProperty`.
- Inverse properties should not be defined (as Literals cannot be subject of statements, inverse does not apply anyway).

For all custom-created properties and classes, modification dates *must* be set.

6.18 Collections of Things

In Personal Information Management, grouping multiple Things into one collection is a crucial feature. Today's hierarchical file systems are a good example:

a folder can be created to contain multiple elements. Later, actions on this folder, such as compressing it, or deleting it are supported. The generic *has Part* relation provides the semantics of putting a Thing into another Thing. For usability reasons, we also provide a class `pimo:Collection` to be used for generic collections of multiple items.

Applications that want to present the complex possibilities of PIMO in a simpler way can offer collections. First, an instance of the class `pimo:Collection` is created. Then, elements are added to the collection using the `pimo:hasPart` relation. A typical application of collections is the list of "Favourites" containing recently used and important resources.

Collections are unordered, the ordering of items inside the collection can be done using alphabetical order, time, geographic location (if they are locatable), or type.

Tags are another simplification, described below in Section 11.7.

6.19 Modeling Associations and Roles in PIMO

Often there is a need to add meta-data about a relation, for example the date of creation of a relation. In RDF, this is typically done using reification, and then adding meta-data about the reified Statement using an instance of the class `rdf:Statement`. A problem with reification is that when using the generic class `rdf:Statement` to represent it, there are no guidelines which properties are now suitable to annotate the statement. More precise sub-classes of Statement would solve this. Another problem is that *n-ary* relations cannot be expressed with triple statements.

In PIMO, **Associations** are used to add metadata about relations and to create n-ary relations. They are entities representing the relation of multiple Things with each other. Each Thing part of an Association is related to the association using the `pimo:associationMember` property or more precise sub-properties of it.

As an example, the fact that Claudia attended a meeting can be expressed using the `pimo:Attendee` role.

```
claudia:AttendsInitialMeetinginBelfast a pimo:Attendee;
  pimo:attendingMeeting claudia:InitialMeetinginBelfast;
  pimo:roleHolder claudia:Claudia.
```

Here, the class `pimo:Attendee` is a sub-class of `pimo:Association` and represents the association as such ("this is an association between a person and a meeting"). The two relations used are sub-properties of `pimo:associationMember` and identify the two Things to relate, the specific relations determine the role taken by each Thing. New sub-classes of association can be created when needed, also new sub-properties of `pimo:associationMember` for more specific roles.

Associations are elements of a user's PIMO and *must* be connected to the user's PIMO with a `pimo:isDefinedBy` relation. Modification dates are to be handled the same way as with Things (see Section 6.10).

7 Connecting PIMO to Information Elements

In the last section, the Things created within a user-PIMO were described. They are to be unique, described with defined ontologies, and ought to be identified well. The next step is to connect these to the files, e-mails, and other Information Elements which exist in the user's PSI. These are ambiguous,

described in various ontologies, and in general more chaotic when compared to the user-PIMO. The crucial point is to use Things as organization scheme to classify and integrate existing data found in a PSI.

The first step is to **connect Things to Information Elements** that represent them. As described above (Section 6.4), the `pimo:groundingOccurrence` and `pimo:occurrence` relations are to be used to connect them. This connection has the semantics of a unification — both the Thing and the Information Element represent the same real-world entity. But the Thing is the unique, static representation that should be used to annotate the entity. Implementations *should not* allow the users to annotate Information Elements directly, instead it is *recommended* to connect the Information Element to a Thing using `pimo:groundingOccurrence` and then annotate the Thing. The rationale is that Information Elements can change their URI, be deleted or moved, and then the annotations may be disconnected from the described resource.

Creating a Thing for each annotated document will result in vast amounts of instances in the sub-classes of `pimo:Document`, as users can likely have access to thousands (sometimes millions) of documents. To navigate effectively through such large structures, PIMO Topics can be used to annotate documents using the `pimo:hasTopic` relation.

How to annotate documents using PIMO is described in Section 11.7.

7.1 Connecting Things and Classes to Folders

Things can also be connected to folders in the file-system to express that these folders contain information related to the thing. Use the `pimo:hasFolder` relation to connect a Thing or a Class with a folder. The semantic meaning of this relation is not formally restricted but open to be used in various ways. For folders connected to Things, it is *recommended* to interpret the content of the folder as "having the Thing as topic". Implementors *may* add a `pimo:hasTopic` relation between the Things inside the folder and the Thing.

For folders connected to Classes, it is *recommended* to interpret the content of the folder as "being an instance of the Class". Implementors *may* add a `rdf:type` relation between the Things inside the folder and the Class.

In all cases, files or other information elements in folders have to be represented as Things first, before further annotation (see Section 11.1).

The property `pimo:hasFolder` *can* be used by implementors to suggest folders for information elements — if an information element is annotated with a `pimo:hasTopic` relation to a topic that is connected to a folder, this is an indication to move the element to the folder, if needed.

7.2 Integrating Facts about Things

The unification of multiple Information Elements into one Thing is also on the level of facts, RDF statements. To answer the question "when did I last communicate with people shown on this picture" can only be answered when facts from multiple sources (e-mails, photos, photo annotations and the user-PIMO) can be queried as one model. The statements about the Information Elements connected to a Thing via `pimo:groundingOccurrence` and `pimo:occurrence` can be **superimposed** to the Thing. The exact rules are given in Section 12.1 and directions how to use them are in Section 11.6.

Through this process, a view on the data is generated. The user can get an overview of all existing data — in an integrated way — and then drill down into specific occurrences. In this view, it is possible that a Thing has multiple classes (as `rdf:type`), one from the level of PIMO ontologies and others from the inte-

grated Information Elements. In the example given in Section 11.6, Dirk Hagemann is inferred to be both a `pimo:Person` and a `pimo:PersonContact`. The two classes are not required to be sub-classes of each other. To get a coherent and meaningful view, the class of the InformationElement (or related resource) *may* be related to a PIMO class using the *pimo:hasOtherConceptualization* relation, as described later in Section 9.

It is not required that the used ontologies are formally aligned and mapped. Rather, it is assumed that the user will be able to interpret the statements based on his knowledge about the data in his PSI.

The details about the integration of facts are given in Section 11.6 on *unification*.

8 PIMO-group level: Group and Domain ontologies

The upper (foundational) level of *PIMO* just makes a few, basic ontological statements about Things which exist on a Semantic Desktop, *i. e.*, Things which are essential in a knowledge worker's mental model. In order to avoid a *cold start problem* [20] with PIMO-based applications, more ontologies defining concepts shared within groups or modeling domains are needed. The user's company and its organizational structure may be such a domain, or a shared public ontology. Classes are refinements of PIMO-Upper, allowing an integration of various domain ontologies via the upper layer.

In the following section, recommendations are given how to model group and domain level ontologies.

9 Extending PIMO

Out of the box, PIMO is kept sufficiently simple and only contains relatively few classes and properties. This was done in order to ensure that the ontology is general enough to apply to almost any relevant domain. However, as soon as the set of pre-existing classes and properties becomes too narrow and confining, it is a very simple matter to extend PIMO and add domain-specific extensions, or map external ontologies onto PIMO. E.g., PIMO can easily be extended to express the organizational structure of the user's workplace, a biological classification system, or to include a PIMO-version of the BibTeX vocabulary. These *domain ontologies* differ from *personalized classes and properties* (see Section 6.17) by the fact that they are not created by the user, but created by a third party for multiple users.

9.1 Refining Elements of PIMO-upper

Creating group–level ontologies is a simple matter of **defining new sub-classes of PIMO-upper classes (see Sect. 6.13) or sub-classing `InformationElement` classes**. If needed, new properties can also be added, which apply to the new classes via domain or range. Importing created *group–level ontologies* into a user's PIMO is described in the next Section (10).

Classes As an example, you may work in the domain of teaching and training, and therefore want to extend PIMO with elements specific for this domain, such

[20] The problem of cold starts is well known in knowledge-based systems: In the beginning a system, such as a shell, has little or no information and therefore doesn't seem to be useful to a new user. Consequently, they are not motivated to invest in using and feeding the system with new information, which again would be a prerequisite for it to be *more* useful. Enter vicious cycle...

as *courses*, *lessons*, *teachers* or *students*. In this case, you would look for existing PIMO-upper classes which could be considered generalizations of your new classes. E.g., a course would be a sub-class of `pimo:ProcessConcept`, training lessons could be a sub-class of `pimo:Meeting`, teachers and students could be sub-classes of `pimo:Person` (or `pimo:Role` — role-based modeling is discussed in Sect. 6.19) and training material a sub-class of `pimo:Document`. Since all pre-existing PIMO-upper classes derive from `pimo:Thing`, all your new classes automatically do as well (except for roles: `pimo:Role` is not a sub-class of `pimo:Thing`).

There could also be cases where no existing PIMO-upper class seems to apply to your new class — in this case, the new class would directly derive from `pimo:Thing`. Consider, e.g., that you want to include the concept of *grades* in your PIMO. There isn't really a good pre-existing superclass for grades in PIMO-upper, so your new `Grade` class would be a direct sub-class of `pimo:Thing`.

Even if there might be a potential superclass, it may be wise to postpone this decision if one isn't completely sure, and instead just sub-class `pimo:Thing`. It is always easier to add a superclass relationship later, rather than make a bad decision now and then have to deal with incorrect data at a later stage.

If needed, the new classes can also have sub-class relationships into other ontologies, such as other NIE-based ontologies or completely different ontologies such as WordNet[21], SUMO[22], or Dolce[23].

Instances In some cases, you will also want to add instances of classes to the ontology you are integrating with PIMO. This makes sense if those instances are shared and used among a many users. An example are actual grades that a teacher might gives their students, such as grades from A–F. Each such grade (`teaching:GradeA`, `teaching:GradeB`, etc.) is an instance of the class `Grade`, but since those instances will be used by all teachers and students, they can become part of the teaching ontology. Similarly, if a school decides to introduce the teaching ontology, they might include instances for all their teachers, students, classes, courses, etc.

Properties New properties can then refer to the new classes via domain or range and thus further specify them. Examples are the relation between courses and teachers/instructors (e.g., `teachesCourse(Teacher, Course)`) or between course material and a course (e.g. `courseMaterialFor(CourseMaterial, Course)`). This example is illustrated graphically in Fig. 5, as well in N3 source code in Fig. 6. This approach is a **typical example of how to integrate domain ontologies for specific application areas into PIMO**.

There are some general guidelines for introducing new properties:

- Properties which connect two `pimo:Things` (or sub-classes) should be defined as sub-properties of `pimo:related`, `pimo:hasTopic`, `pimo:isTopicOf`, `pimo:hasPart`, or `pimo:partOf` (see Sect. 6.15). By relating new, specialized properties to the more generic PIMO properties, the new ontology can integrate better with existing desktop environment. When not extending the generic properties, at least new properties should exted `pimo:objectProperty`.
- All new object-properties *must* define an inverse property, as required in Section 6.17.
- Identifying properties (such as a name) that have domain `pimo:Thing` and a literal range should be mapped as sub-properties of `nao:identifier`.

[21] `http://wordnet.princeton.edu/`
[22] `http://www.ontologyportal.org/`
[23] `http://www.loa-cnr.it/DOLCE.html`

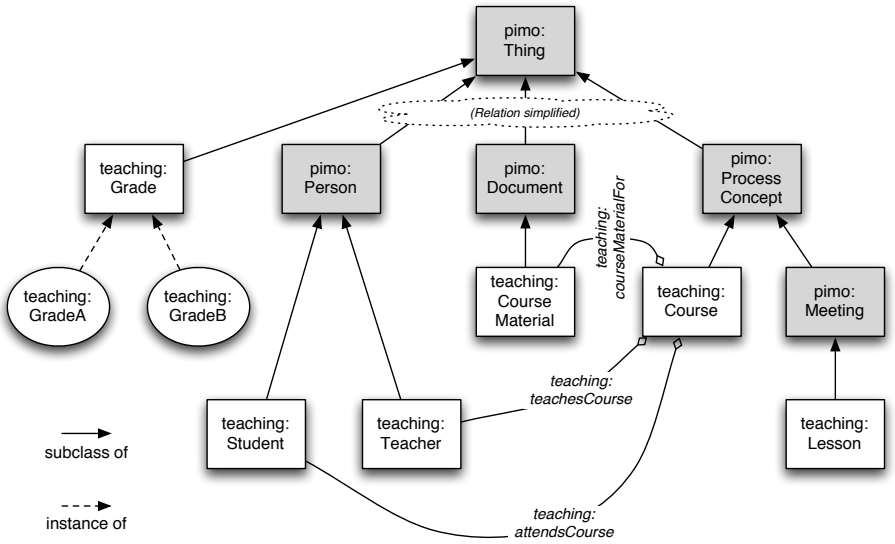


Figure 5: Extending PIMO with new classes, properties and instances for the domain of teaching

An example is give in Fig. 8.

- Some new properties may be defined as sub-properties of `pimo:referencingOccurrer` (see Sect. 6.4). This is true for all object properties (i.e., properties which have a resource range and not a literal range) which describe or identify the subject in an unambiguous way. In other words, the object resource exclusively describes the subject. A typical example is `foaf:homepage`: two different people would most likely not have the same homepage (ignoring exceptions such as family homepages). If, however, we come across two different RDF resources which have the same `foaf:homepage`, we can assume that they describe the same real-life person.
- A frequent situation in Semantic Web and Semantic Desktop scenarios is that the same real-life object (i.e., a person, country, project, etc.) is defined as a resource in many different ontologies. The PIMO property `pimo:hasOtherRepresentation` is used in such cases. If your new ontology contains a property which expresses a similar (more specific) relation between resources, then it should be a sub-property of `pimo:hasOtherRepresentation` (see Sect. 6.4). In the vanilla SW world, a similar property is `rdfs:seeAlso`.

Inheritance Sub-classing any class from PIMO (whether it be existing classes from PIMO-upper or classes that have been added later) also means that the new sub-class can be used with the same properties that have been defined with its superclass. Remember that NRL has a closed world assumption, and not an open world assumption, as RDFS traditionally has. In NRL[24], ontologies can be used to validate statements. E.g., if the property `name(pimo:Person, String)` has been defined, and if we define our new class `teaching:Student` to be a sub-class of `pimo:Person`, then this will allow us to use `name` with instances of `Student` as well - an NRL validator will permit this, because all instances of `Student` are also instances of `Person`. An example of this is shown in Fig. 7: this concept is very similar to the idea of inheritance in object-orientation, even though strictly speaking it is not the same.

[24] `http://www.semanticdesktop.org/ontologies/nrl/`

```
# new classes:
   teaching:Grade a rdfs:Class;
      rdfs:subClassOf pimo:Thing.

   teaching:Student a rdfs:Class;
      rdfs:subClassOf pimo:Person.

   teaching:Teacher a rdfs:Class;
      rdfs:subClassOf pimo:Person.

   teaching:CourseMaterial a rdfs:Class;
      rdfs:subClassOf pimo:Document.

   teaching:Course a rdfs:Class;
      rdfs:subClassOf pimo:ProcessConcept.

   teaching:Lesson a rdfs:Class;
      rdfs:subClassOf pimo:Meeting.

# new properties and their inverse:
   teaching:courseMaterialFor a rdf:Property;
      rdfs:subPropertyOf pimo:partOf;
      rdfs:domain teaching:CourseMaterial;
      rdfs:range teaching:Course;
      nrl:inverseProperty teaching:hasCourseMaterial.
   teaching:hasCourseMaterial;
      rdfs:subPropertyOf pimo:hasPart;
      rdfs:domain teaching:Course;
      rdfs:range teaching:CourseMaterial;
      nrl:inverseProperty teaching:courseMaterialFor.

   teaching:teachesCourse a rdf:Property;
      rdfs:subPropertyOf pimo:related;
      rdfs:domain teaching:Teacher;
      rdfs:range teaching:Course;
      nrl:inverseProperty teaching:taughtBy.
   teaching:taughtBy a rdf:Property;
      rdfs:subPropertyOf pimo:related;
      rdfs:domain teaching:Course;
      rdfs:range teaching:Teacher;
      nrl:inverseProperty teaching:teachesCourse.

   teaching:attendsCourse a rdf:Property;
      rdfs:subPropertyOf pimo:related;
      rdfs:domain teaching:Student;
      rdfs:range teaching:Course;
      nrl:inverseProperty teaching:attendeeStudent.
   teaching:attendeeStudent a rdf:Property;
      rdfs:subPropertyOf pimo:related;
      rdfs:domain teaching:Course;
      rdfs:range teaching:Student;
      nrl:inverseProperty teaching:attendsCourse.

# new instances:
    teaching:GradeA a teaching:Grade;
      nao:prefLabel "A".

    teaching:GradeB a teaching:Grade;
      nao:prefLabel "B".

    ...
```

Figure 6: Extending PIMO with new classes, properties and instances for the domain of teaching — N3 code

```
pimo:name a rdf:Property;
   rdfs:Domain pimo:Person;
   rdfs:Domain rdfs:Literal.

teaching:Student a rdfs:Class;
   rdfs:subClassOf pimo:Person.

knud a teaching:Student;
   pimo:name "Knud Möller".
```

Figure 7: “Inheritance” of properties

9.2 Markup for the new ontology

The teaching ontology still needs to be defined as proper NRL ontology to be usable with PIMO. The ontology *must* to be identified via its URI and the author of the ontology can be added as `nao:creator`.

```
teaching:TeachingOntology a nrl:Ontology;
 nao:creator teaching:TeachingOntologyCreator.

teaching:TeachingOntologyCreator a nao:Party;
 rdfs:label "Knud Möller".
```

9.3 Information Elements

An important feature of the NEPOMUK ontology architecture is the fact that it is divided into two tiers: the PIMO tier and the Native Structures tier, as defined in the NEPOMUK Information Element Ontology (NIE)[25] and its sub-ontologies, such as the NEPOMUK File Ontology (NFO)[26]. While the former covers the internal mental model of a user or an organization (people, events, projects, etc.), the latter covers the physical representations of data (address book entries, calendar entries, files, etc.). Obviously, there are numerous connections between the tiers: people are represented by address book entries, events appear in the calendar, projects have files associated to them.

Whenever classes that are introduced to PIMO have a physical representation on the user’s desktop, a connection to NIE must be modeled as well. Consider the example of the teaching ontology above: Such an ontology will contain classes for Things like exams and essays. Those classes belong to the PIMO tier. Their representation within the computer — e.g., as text files — belongs to the native structures tier. Or, in a more complex case, the new ontology could very well come with a specialized application, such as a Training Course Manager, where users can assign attendees to trainings, etc. In this case, people and courses (PIMO tier) would be represented by the application as application-specific data structures or information elements (native structures tier). In both cases a link from the new PIMO classes to the information elements represented with NIE is required to fully exploit the possibilities of the semantic desktop.

As a second example, we can consider an ontology for scientific publications. This ontology (which would probably be based on BibTEX), would come with classes such as `Article` or `Book` and relations between them like `bookContainsArticle` or `hasAuthor`. For the integration into PIMO, `Article`

[25] `http://www.semanticdesktop.org/ontologies/nie/`
[26] `http://www.semanticdesktop.org/ontologies/nfo/`

```
bibtex:Article a rdfs:Class;
 # PIMO tier: interpreted by the user as nie:Document
 rdfs:subClassOf pimo:Document;
 # native structures tier: interpreted by the system as nfo:TextDocument
 rdfs:subClassOf nfo:TextDocument.

bibtex:hasAuthor a rdf:Property;
 rdfs:subPropertyOf pimo:related;
 rdfs:subPropertyOf nfo:creator;
 rdfs:domain bibtex:Article;
 rdfs:range pimo:Person.

bibtex:hasLCCN a rdf:Property;
 rdfs:subPropertyOf nao:identifier; \# add an identifier
 rdfs:domain bibtex:Article.
```

Figure 8: A BibTEX-based PIMO extension for scientific publications

would most likely become a sub-class of `pimo:Document`. Documents have two types: one which anchors them in the PIMO tier, and one which anchors them in the native structures tier. The `pimo:LogicalMediaType` (and its sub-classes, e.g., `pimo:Document`) captures how a document is interpreted by the user and belongs to the PIMO tier. Logical media types can be contracts, invoices, assignments, invitations, law texts, etc. The other type, which is the `nfo:Document` type, captures how the system interprets the document, and belongs to the native structures tier. This is the physical document type as modeled by NIE. Instances of a logical media type can have various representations in the native structures tier: for example, a text interpreted as an invoice by the user can either be an `nfo:PlainTextDocument` or an `nfo:PaginatedTextDocument`. Vice-versa, one physical type can be used to represent both an invitation or an invoice, which are different logical media types. Keeping this this separation of content and representation in mind, one can model concrete documents having two types, one on each tier.

9.4 Extension by Sub-classing from External Classes

Another possibility is extending existing PIMO-upper classes by sub-classing them from external classes. *This is discouraged*. For example, if the class `pimo:Person` was defined a sub-class of `nco:PersonContact` and `pimo:Organization` a sub-class of `nco:OrganizationContact`, all instances of these classes would automatically be inferred to be `pimo:Things`. However, those instances would probably not have some of the properties required by the `Thing` defined, which would render the imported data invalid. Similarly, when a mapped class X has cardinality restrictions on its properties (such as required properties), adding X as new superclass to an existing PIMO class can render the instances of the PIMO class invalid.

9.5 Summary

The very condensed summary to extending PIMO and mapping frome existing ontologies to PIMO is the following:

- Make classes sub-classes of PIMO-upper classes.
- Make properties sub-properties of PIMO-upper properties.

- Relations: Links between Things have to be browseable, properties should have inverse relations defined (see [16])
- Extensibility: Users are free to add new relation types and new classes (see [16])

10 Importing Domain Ontologies into a User's PIMO

Once modeled, the new domain ontology such as the teaching ontology in the previous examples can be made available publicly for others, for example by publishing it on the Web. A good reference for doing this is *Best Practice Recipes for Publishing RDF Vocabularies* [12].

Semantically, an imported ontology is captured using a `nrl:imports` statement. When a user imports a domain ontology, this statement *should* be added.

```
claudia:Pimo nrl:imports teaching:TeachingOntology.
```

Once the user of a Semantic Desktop system imports an external PIMO into their own desktop, all new classes (which are sub-classes of `pimo:Thing`) *should* become available to them as if they had created those classes themselves. Instances, on the other hand, are not automatically available. As said in the introduction, the scope of a PIMO for an individual user is to model data that is within their own attention and needed for knowledge work or private use. However, when importing external ontologies or knowledge bases, not all instances may be of interest to the user. As with imported information elements, a separate Thing is created to represent the imported resource within the PIMO of the user and connected to the imported instance using `pimo:hasOtherRepresentation`[27]. Before creating a new Thing for an imported instance, the PIMO of the user has to be checked if the entity is already represented as a Thing, as indicated above in Section 11.1. Once represented as a Thing in the user's PIMO, it is possible to assign a personal identifier to it, annotate it, and use it.

Implementations *must* automate the importing process. The user should be able to interact with imported Things as if they were created by themselves.

11 Practical Directions on Using PIMO

In this section, a few issues that will arise in actual PIMO usage are discussed. For each of these issues, we will suggest a recommended way of handling them. Even though those things are not strictly speaking part of the ontology, they are part of the standard defining how to use the PIMO ontology in applications. Implementers *should* conform to these directions.

11.1 Creating Things

New PIMO Things can be created freely, but usually the creation of a Thing is rooted in the existence of an information element. An algorithm to create new Things based on information elements *should* follow these steps:

[27] An alternative would be to treat all Things present in the RDF data available of the user as if they were created by the user, imported or not. This has the drawback that it allows to represent the same real-world entity twice, first in the imported domain ontology and second in the user's own PIMO.

Start The software agent encounters a resource with URI X and wants to verify if the user already has knowledge about X.

Check GroundingOccurrence Query the user-PIMO for:

```
SELECT ?thing WHERE {?thing pimo:groundingOccurrence ?X.}
```

When a Thing is found, finished.

Check occurrences Repeat the last step to search if X is an occurrence of a Thing.

Check identifiers Validate if the InformationElement has an identifier or a referencing occurrence that is also used on an existing Thing. The information element is called an **occurrence** of a Thing when it shares the same identifiers. The correct query is:

```
SELECT ?thing
WHERE {
?thing ?p ?o.
?X ?p ?o.
?p rdfs:subPropertyOf nao:identifier.
} UNION {
?thing ?p ?o.
?X ?p ?o.
?p rdfs:subPropertyOf pimo:referencingOccurrence.
}
```

When a Thing is found, finished.

Create a new Thing When the last step did not return an existing Thing, this can be an indicator that element X is new to the user and should be modeled with a new Thing. Mint a new URI and add the identity values from the InformationElement.

In the following SPARQL query example, we assume that Claudia's System has just encountered a new calendar event X and represents it using the new minted URI $claudia : Event42$.

```
CONSTRUCT {
<claudia:Event42> rdf:type pimo:Thing.
<claudia:Event42> ?i ?io.
<claudia:Event42> nao:prefLabel ?title.
<claudia:Event42> rdf:type  ?type.
<claudia:Event42> rdf:type  ?pimotype.
<claudia:Event42> pimo:groundingOccurrence ?X.
<claudia:Event42> nao:created ‘‘2007-06-30T18:11:00Z’’.
<claudia:Event42> pimo:isDefinedBy claudia:PIMO.
} WHERE {
OPTIONAL (?X ?i ?io. ?i rdfs:subPropertyOf nao:identifier.).
OPTIONAL (?X nie:title ?title).
OPTIONAL (?X rdf:type  ?type).
OPTIONAL (?X rdf:type  ?type. ?pimotype rdfs:subClassOf ?type.
 ?pimotype rdfs:subClassOf pimo:Thing).
}
```

The different parts of this query are:

- all identifiers are copied (?i, ?io)
- the title is copied for readability and to use it for tagging (?title)

- the original type(s) are copied (?type)
- find possible PIMO:Thing classes that can be used for this type (?pimo-type)
- the groundingOccurrence relation is added
- the created timestamp is set to the current date
- the isDefinedBy relation is set

Finding a suitable class to represent the resource in the PIMO depends on a mapping of information element classes to PIMO classes and can be realized in different ways, see Section 6.11.

11.2 Changing the Type of a Thing

Changing the type of an instance can lead to invalid statements in the knowledge base. E.g., this can happen when the instance in question is involved in statements using a property with a specific domain and range, and the new type is not compatible to those. As this would render the model invalid, this situation must be avoided in applications interacting with PIMO data. A user interface should remind the user of possible problems and should suggest solutions.

A different approach is ontology-by-examples where the main principles is that "the user is always right" and domain/range values are not explicitly set but implied by previous usage. User interfaces may support this behavior by setting the domain and range to `pimo:Thing` and suggest properties suitable for class by analysing previous usage of the property.

11.3 Deleting a Thing

We can say that, in RDF, instances do not exist independently, but only as part of statements. Thus, "deleting an instance" really means deleting all statements in which this instance is involved (as subject or object). If the instance has a grounding occurrence relation, a new `pimo:groundingForDeletedThing` relation should be created to record the deletion. Data integration algorithms analysing resource to automatically create Things can then avoid re-creating the already deleted Thing.

11.4 Deleting User-generated Classes and Properties

Deleting a user-generated class is allowed but requires that instances are re-typed to the direct super-class of the deleted class. Also the domains and ranges of all properties defined with the class as domain or range need to be changed to the direct superclass. These changes are necessary to prevent the model from becoming invalid. Classes defined in the pimo language cannot be deleted, only classes defined by the user can be deleted.

The direct super-class S of the class C is defined as the class where no other class T exists where C has superclass T and T has superclass S. If there are multiple S, all of them should be used as replacement when deleting a class.

If user-generated properties are deleted, this requires all statements in which those properties are involved to be deleted as well, in order to prevent the model to become invalid. Alternatively, all occurrences of the property could be replaced by its super-property.

11.5 Merging Duplicates

Duplicates are two Things that represent the same real-world entity within a user's PIMO. As we apply the unique name assumption (two Things with different names are different), duplicates should be found and corrected. When duplicates are found, both (the "duplicate" and the "original") can be merged into one single instance. In this process, RDF statements involving the duplicate are changed so that they now refer to the original. In a local setup with one desktop and all data stored in a single database, this process is without information loss and causes no side-effects. As Semantic Web applications are in a distributed scenario, deleting a duplicate resource can cause dangling links in related databases and other side-effects.

For this, the `pimo:hasDeprecatedRepresentation` property *should* be used to relate the original with the (now deleted) duplicate. Note that the range of this property is not defined, as all data about the duplicate resource (including the type) is deleted.

When merging two duplicate resource A and B into one resource C, a few practical guidelines can reduce side-effects:

- The URI of C is either A or B, the older resource (as defined by `nao:creationDate`, see Section 6.10) is to be preferred as it has a higher chance of being used.
- The classes of C are the union of the classes of A and B.
- All statements about A and B are merged to C.

11.6 Unification of multiple Information Elements into one Thing

In comparison to merging duplicate Things, *unification* is the process when multiple information elements representing the same real-world entity are mapped to one `pimo:Thing` instance.

In PIMO, multiple information elements representing one real-world entity are mapped to exactly one `pimo:Thing` instance using the `pimo:groundingOccurrence` and `pimo:occurrence` relations. Algorithms or user interfaces implementing unification must consider the identifying properties of a Thing (see Section 6.4) when searching for possible Things to representing information elements.

Four default **NRL Views** and **NRL ViewSpecifications** are defined for different levels of inference. Each creates a named graph that contains the instance data and the inferred statements.

`pimo:InferOccurrences` a view that infers occurrences based on `nie:identifiers` and pimo:referencingOccurrence annotations.

`pimo:GroundingClosure` a view that adds statements about the grounding Occurrences and `hasOtherRepresentation` to a Thing.

`pimo:OccurrenceClosure` a view that adds statements about all occurrences to a Thing.

`pimo:FullPimoView` a supergraph of all above.

By providing these graphs, we let the user and software agent decide if the full closure is needed at all times. When no closure is needed, the plain NRL data graphs can be used as-is. To answer complex queries like "Which e-mails were sent to me by attendees of meetings that I have today", the full closure is a good choice.

The ability to superimpose data using inference limits the data needed in a PIMO to a necessary minimum: only the identification properties are mandatory, the occurrence and the hasOtherRepresentation properties superimpose existing data.

```
claudia:DirkHagemann a pimo:Person;
 pimo:occurrence <imap://claudia@example.com/INBOX/1#from>;
 pimo:groundingOccurrence <file://home/claudia/dirk.vcf#dirk>.
```

Using the guiding example, an integrated view of Claudia on Paul is the following, assuming full closure:

```
# The canonical Dirk
claudia:DirkHagemann a pimo:Person;
# the second type is also inferred
 a nco:PersonContact;
 pimo:isDefinedBy claudia:PIMO;
 nao:prefLabel 'Dirk Hagemann';
 nao:identifier "dirk@example.com";
 pimo:occurrence <imap://claudia@example.com/INBOX/1#from>;
 pimo:groundingOccurrence <file://home/claudia/dirk.vcf#dirk>;
 pimo:referencingOccurrence <http://www.example.com/people/DirkHagemann>;
 pimo:hasOtherRepresentation <http://id.example.com/person/1650>;
# the inferred facts
 nco:hasEmailAddress <mailto:dirk@example.com>;
 nco:nameFamily "Hagemann";
 nco:nameGiven "Dirk";
 nco:photo <http://www.example.com/people/dirk/photo.jpg>.

# E-mail, now pointing to the canonical Dirk
<imap://claudia@example.com/INBOX/1> a nmo:Mail;
 nmo:from claudia:DirkHagemann.
```

11.7 Tagging and Annotating Files

Conceptually, once an Information Element (a file, an e-mail, a webpage, an address book entry, etc.) is in the attention of the user and is read or annotated, it is also a Thing in the mental model of the user. A core idea of the Semantic Desktop is to use other Things as dimensions to annotate files and retrieve them later, as described in [5].

Lets assume the file `/home/claudia/doc/tripplan.pdf` is to be tagged with the tag “Belfast Meeting Package”.

A representation of the file is already extracted by the semantic desktop data wrapper system (or any other content management system):

```
<file://home/claudia/doc/tripplan.pdf> a nie:TextDocument;
 nie:language "en";
 nie:title "Belfast Bus Timetable".
```

Additionally, the city of Belfast and the Belfast meeting package already exist in the model, as well as a `MeetingPackage` class (which was either created by Claudia, or came as part of a shared domain/group ontology):

```
claudia:BelfastMeetingPackage a claudia:MeetingPackage;
 nao:personalIdentifier "Belfast Meeting Package";
 pimo:isDefinedBy claudia:PIMO.
```

To add a simple tag, the file is now represented as a Thing (*thingified*) and the tagging relation set.

```
claudia:BelfastBusTimetable a pimo:Document;
  nao:personalIdentifier "Belfast Bus Timetable";
  pimo:isDefinedBy claudia:PIMO;
  pimo:groundingOccurrence <file://home/claudia/doc/tripplan.pdf>;
  nao:hasTag claudia:BelfastMeetingPackage.
```

The interested reader may now ask, "but why create a new Thing and not just add the nao:hasTag relation directly to the file?" The reason to *thingify* the file is twofold: first, it is possible to assign a new class to the file, for example creating the class `claudia:BusTimetable`. Second, the same timetable may be available in different data objects, if it is stored in an e-mail attachment, a web resource, or a local file — it is always the same document. A tag added to one occurrence of the file is also valid for other occurrences. Also refer to Sections 7,9.3.

Given Claudia finds the same timetable on a website, the system can link the two based on `nie:identifiers` (which should be globally unique identifiers for information elements, independent of the data object they are stored in). This example shows this case (with copied identifiers):

```
# the file on the harddisk
<file://home/claudia/doc/tripplan.pdf> a nie:TextDocument;
 nie:language "en";
 nie:title "Belfast Bus Timetable";
 # this isbn is fictional
 nie:identifier "ISBN:12123-123123".

<http://www.buseireann.ie/timetables/belfast.pdf>
 a nie:TextDocument;
 nie:title "Belfast Bus Timetable";
 # this isbn is fictional
 nie:identifier "ISBN:12123-123123".

claudia:BelfastBusTimetable a pimo:Document;
  pimo:isDefinedBy claudia:PIMO;
  pimo:groundingOccurrence <file://home/claudia/doc/tripplan.pdf>;
  pimo:occurrence <http://www.buseireann.ie/timetables/belfast.pdf>;
  nie:identifier "ISBN:12123-123123";
  nao:hasTag claudia:BelfastMeetingPackage.
```

The tag assigned to the bus time table is now valid, independent of the data object.

Using the generic tag relation **nao:hasTag** is an easy-to-use entry point for users that don't need formal semantic relations.

Instead of using the `nao:hasTag` relation, one of the generic PIMO relations (`related`, `partOf`, `hasTopic`) could be used, see Sect. 6.15 or properties defined in domain ontologies.

Example: The Belfast Bus Timetable has the topic Belfast, which is not just a Tag but says more about the relation between the timetable and the city. The **has Topic** relation can be used for this:

```
claudia:BelfastBusTimetable a pimo:Document;
 pimo:hasTopic claudia:Belfast.
```

Example: The relation between Claudia and the BelfastMeetingPackage can be expressed using "has tag" but may also be represented with a relation saying more about the semantics: Claudia is "attending this meeting". Assuming that the MeetingPackage is a sub-class of meeting, this could be expressed like:

```
claudia:BelfastMeetingPackage a meeting:MeetingPackage.
```

```
# instead of nao:hasTag, a semantic relation is used
claudia:Claudia pimo:attendee claudia:BelfastMeetingPackage.
```

11.8 Geo-locating Things

Things can be geo-located, meaning that their geographical location is set using latitude, longitude and altitude information in the WGS84 geodetic reference datum. In PIMO, the location is a separate entity of type `Location`, and other items can then be geo-located there. The relation `pimo:hasLocation` is used for this, `pimo:isLocationOf` is the inverse. There is an abstract marker class for locatable Things: pimo:Locatable. Social events, organizations, persons, and physical objects are locatable.

To add a location to a meeting, this data is added, for your reference the location is also given:

```
claudia:InitialMeetinginBelfast a pimo:Meeting;
  nao:prefLabel "Initial Meeting in Belfast";
  pimo:hasLocation claudia:Belfast.

claudia:Belfast a pimo:City;
  nao:prefLabel "Belfast";
  geo:lat "54.5833333";
  geo:long "-5.9333333".
```

Usually, locations should be reused to locate multiple Things, but new locations can of course be generated anytime.

PIMO defines a number of general-purpose classes for countries, states, cities, buildings, and rooms, which we consider independent of culture and domain. Domain-specific ontologies can specify those classes further. In the rare case when the type of location cannot be specified with any of the existing classes, the superclass `pimo:Location` can be used. In some application scenarios (such as geo-locating a large amount of photos or measurement values), many locations would be needed. To simplify annotation and remove clutter, specialized vocabularies may then be used, as for example done in the NEXIF vocabulary for photos.

11.9 Defining what is in the PIMO and what is not: NRL Graphs and `definedBy`

The facts (i.e., the statements) used to describe Things should be kept in named graphs according to the NRL standard. This allows to express metadata about the facts, such as to which PIMO the Things belong (see Sect. 6.3) or when individual triples were changed (see Sect. 6.10) and by whom. Another important feature is to find which Things are in the user's PIMO and which not. In a shared environment, users may often import group-level ontologies that contain Things modeled by others. Considering this, it is important to keep up-to-date as to who said what, or, in other words, to detect if a Thing was modeled by the user or not.

All data expressed about Things can be kept in NRL graphs. The minimal metadata is:

- The type of the graph is `nrl:KnowledgeBase`, as it can contain both instances and ontology constructs (see the NRL specification for more details).
- The graph is imported into the user's PIMO.

Additional metadata such as `nao:created` can be added. These facts can be expressed in a separate metadata graph. An example of how this looks like in code using Trig Notation[28]:

```
# The instance data
claudia:graph1 {
  claudia:DirkHagemann a pimo:Person;
    nao:prefLabel 'Dirk Hagemann'.
}
# The metadata
claudia:graph1_metadata {
  claudia:graph1 a nrl:KnowledgeBase;
  claudia:PIMO nrl:imports claudia:graph1.
}
```

As not all implementers can or want to use NRL graphs to keep track of the boundaries of a PIMO, `pimo:isDefinedBy` **must be used** to connect classes, properties, and instances to the PIMO that defines them[29].

To support both ways of detecting the boundaries of a user-PIMO, implementations *may* add the NRL metadata in addition to the `pimo:isDefinedBy` statements to the generated data. Resulting in:

```
# The instance data
claudia:graph1 {
  claudia:DirkHagemann a pimo:Person;
    pimo:isDefinedBy claudia:PIMO.
    nao:prefLabel 'Dirk Hagemann'.
}
# The metadata
claudia:graph1_metadata {
  claudia:graph1 a nrl:KnowledgeBase;
  claudia:PIMO nrl:imports claudia:graph1.
}
```

11.10 Using NAO and NIE Elements for Annotation

Throughout this document, we have shown several uses of NAO and NIE elements that can be applied to `pimo:Thing` instances.

Semantically, we start from the assumption that a Thing modeled in PIMO is describing a concept from the real world. On the other hand, the Thing itself can be viewed as a set of statements or triples stored in a computer system. Thus, Things are resources and can be annotated with NAO. Also, using PIMO closure as described in Sect. 7.2, facts of NIE elements can be inferred as facts of PIMO Things. Thus, Things can also have a NIE class as type.

In Table 1 we give an overview of how the semantics of the predicates may alter when interpreted within the PIMO.

The text "**no change**" means that the interpretation is unchanged, "**not interpreted**" means you should avoid interpreting this property in PIMO. You may notice that some properties are not interpreted because they have DataObjects or Strings as range, where PIMO has properties modeling the same semantics but using other `pimo:Things` as range, which is a more precise way of modeling because of the UNA. "**Required**" means that this property is required for all instances of `pimo:Thing` and this can be validated using the rules from Section

[28] `http://sites.wiwiss.fu-berlin.de/suhl/bizer/TriG/`

[29] Separation based on the namespace was considered by some to be enough, but is not (as stores use fulltext comparison to handle namespaces, namespaces as such don't exist in RDF but only in XML).

12.2. “**Recommended**” means that you should use this property as if it is required, but it cannot be validated in NRL as property restrictions are only available in OWL.

The properties describing how information is stored (mime-type, bytesize, characterset) are obsolete for Things, as they describe a serialization.

11.11 How to Infer Knowledge Using Rules?

The presented ontology may be used to express facts that can be used to infer new knowledge. For example, the person Claudia may be working on the project “CID”. Claudia is also employed by the company Example Inc. — this may be used to infer that the company Example is involved in “CID” Such rules are very useful to improve the results when querying, as they will provide more answers.

In PIMO and NRL, it is not straightforward how to express these rules. Also, as PIMO-upper is generic, any rules defined on this level may not apply in all group-level ontologies.

We recommend to express such rules using a rule language (such as the ones defined by the RIF working group of W3C[30]) or SPARQL-create statements. These should be part of domain and group ontologies.

12 Rules Defined by PIMO

Some of the rules defined in PIMO cannot be expressed using ordinary NRL semantics. These rules are written here using SPARQL construct queries **and are also part of the ontology, by virtue of** `nrl:RuleViewSpecification`. The rules help to infer additional statements or validate a model. **All rules in this section assume that at least the sub-class and sub-property relations are inferred.**

12.1 Construction Rules

The first set of rules define new information that is inferred from existing data. They are used for integrating facts about Things by blending, see Sect. 7.2. The rules are expressed based on the assumption that Tbox inferenced facts are available in querying, whereas Abox inference may or may not. This means, sub-class and sub-property relations are fully inferred, but the statements implied by them may or may not[31]. For some of the rules, the semantics of equality comparison is clearer under this assumption. For example, assume two sub-properties of `nao:identifier`: `book:number` and `person:number`, both with a range of integer and values starting with numbers from 0. Expressing a rule as “two Things are equal when their `nao:identifier` is equal” could imply, that a book is equal a person.

`pimo:InferOccurrences` **Level**: These are the rules to infer what resources are occurrences of Things based on identifiers. You can use this approach to integrate data from large stores. For example to see all appearances of a document using the document’s NIE-identifier, this query searches for the `pimo:occurrence` relations.

```
CONSTRUCT {
```

[30] `http://www.w3.org/2005/rules/`

[31] In instance bases, rules rdfs7 and rdfs9 make the majority of inferred statements and for storage optimization `http://www.w3.org/TR/rdf-mt/#RDFSRules`

Term	Semantics
nao:created	**Recommended** When the RDF Resource representing this Thing was first created in the user's PIMO.
nao:identifier	no change, used for identification and matching.
nao:lastModified	**Recommended** When the Resource representing this Thing was last changed in the user's PIMO. Note that it can be different from the nie:contentLastModified property of a groundingOccurrence. When blending multiple groundingOccurrences into a Thing, the latest contentLastModified should be chosen.
nao:modified	**Recommended**
nao:personalIdentifier	**Recommended**
nao:prefLabel	**Recommended**
nie:characterSet	**Not interpreted**, obsolete in RDF
nie:comment	no change
nie:contentCreated	**Not interpreted**
nie:contentLastModified	**Not interpreted**
nie:contentSize	**Not interpreted**, obsolete in RDF
nie:copyright	no change, can be set by the user when sharing Things with others. See also nie:legal
nie:depends	no change
nie:description	no change
nie:disclaimer	**not interpreted**
nie:generator	no change, pimo:Things can be generated automatically by software or manipulated by user interfaces
nie:generatorOption	no change
nie:hasPart	**not interpreted**, pimo:hasPart should be used
nie:identifier	no change
nie:isStoredIn	**not interpreted**, pimo:isDefinedBy covers the storage.
nie:keyword	**not interpreted**, should be modeled using hasTopic and assigning the keyword as title of the topic.
nie:language	no change
nie:legal	no change, by default all legal properties scope all the statements having the same subject as the legal statement
nie:license	no change
nie:licenseType	no change
nie:links	**not interpreted**, use pimo:related or pimo:hasTopic
nie:mimeType	**not interpreted**, obsolete in RDF
nie:plainTextContent	no change
nie:relatedTo	**not interpreted**, use pimo:related
nie:subject	**not interpreted**, use pimo:hasTopic
nie:title	**not interpreted**
nie:version	no change

Table 1: Semantics of NAO and NIE in PIMO

```
?thing pimo:occurrence ?occurrence.
} WHERE {
?i rdfs:subPropertyOf nao:identifier.
?thing ?i ?value.
?occurrence ?i ?value.
}
```

`pimo:GroundingClosure` **Level**: This adds all facts from grounding occurrences and `hasOtherRepresentation` occurrences to Things.

```
CONSTRUCT {
?thing ?p ?o
} WHERE {
?thing pimo:groundingOccurrence ?x.
?x ?p ?o.
}
CONSTRUCT {
?s ?p ?thing
} WHERE {
?thing pimo:groundingOccurrence ?x.
?s ?p ?x.
}
CONSTRUCT {
?thing ?p ?o
} WHERE {
?thing pimo:hasOtherRepresentation ?x.
?x ?p ?o.
}
CONSTRUCT {
?s ?p ?thing
} WHERE {
?thing pimo:hasOtherRepresentation ?x.
?s ?p ?x.
}
```

`pimo:OccurrenceClosure` **Level**: This adds all facts of all occurrences to Things.

```
CONSTRUCT {
?thing ?p ?o
} WHERE {
?thing pimo:occurrence ?x.
?x ?p ?o.
}
CONSTRUCT {
?s ?p ?thing
} WHERE {
?thing pimo:occurrence ?x.
?s ?p ?x.
}
```

A broader Topic is also the Topic of a Thing. If a Thing X has the topic A and topic A has a broader topic B, then X has also the topic B. `broader` and `narrower` are transitive.

```
CONSTRUCT {?x pimo:hasTopic ?B}
WHERE
{?x pimo:hasTopic ?A.
?A pimo:broader ?B.}
```

Note that this is not true for the `hasPart` relation.

An alternative to the `hasTopic` rules would have been to represent topics as RDFS classes (instead of having them as instance of type `pimo:Topic`) and using `rdfs:subClassOf` relations instead of broader/narrower. But this has a nasty side-effect that for a topic like “Web” having a sub-topic “Semantic Web”, the user would suddenly be able to create instances of “Semantic Web”, this would be confusing.

12.2 Validation Rules

These rules validate a model. Some assumptions stated in the text can be validated using these rules. As a model for validation, we have chosen to pick a similar approach as the Jena validation engine, namely creating errors. The classes `error:Error` and `error:Message` were introduced for this purpose and not defined any further, we assume Errors can have parameters that are passed back as `error:param1`, `error:param2`, etc. The parameters can be referenced in the error message for readability.

12.3 Rules Valid when Integrating with NIE

When using PIMO in coordination with NIE, certain properties from NIE can be reused. The rules are often used to restrict properties defined in the NIE ontology to be mandatory on `pimo:Things`, whereas they are optional when used on `nie:InformationElement`. Such restrictions are possible in OWL but not in NRL.

Every Thing must have a `nao:prefLabel`.

```
CONSTRUCT {
_:err a error:Error.
_:err error:Message ‘‘Thing \%1 does not have a nao:prefLabel’’.
_:err error:param1 ?x.
} WHERE {
?x rdf:type pimo:Thing.
OPTIONAL { ?x nao:prefLabel ?title } .
FILTER (!bound(?title))
}
```

Every Thing must have `nao:created`.

```
CONSTRUCT {
_:err a error:Error.
_:err error:Message ‘‘Thing \%1 does not have a nao:created’’.
_:err error:param1 ?x.
} WHERE {
?x rdf:type pimo:Thing.
OPTIONAL { ?x nao:created ?created } .
FILTER (!bound(?created))
}
```

13 Sources considered for designing PIMO

Several ontologies and scientific publications predate this specification. A definition of the term PIMO was given in [17]: A PIMO is a Personal Information Model of one person. It is a formal representation of parts of the users Mental Model. Each concept in the Mental Model can be represented using a Thing

or a sub-class of this class in RDF. Native Resources found in the Personal Knowledge Workspace can be categorized, then they are occurrences of a Thing.

A similar approach was used by Huiyong Xiao and Isabel F. Cruz in their paper on “A Multi-Ontology Approach for Personal Information Management”, where they differentiate between *Application Layer, Domain Layer and Resource Layer*. Alexakos et al. described “A Multilayer Ontology Scheme for Integrated Searching in Distributed Hypermedia” in [3]. There, the layers consist of an *upper search ontology layer, domain description ontologies layer, and a semantic metadata layer*.

PIMO is different from Topic Maps (TM) in that it is based on the logical and semantic foundations of RDF and RDFS, whereas TM have no such foundation. A major difference between TM and RDF is that Topic Maps Associations are n-ary relations, whereas in RDF relations are always binary. In RDF, a similar approach as to TM is the SKOS vocabulary [7]. It represents all Things using the class *Concept*, this prohibits reusing inference and typed properties of concepts (e.g., the “first name” property of a person cannot be modeled in SKOS).

The idea of mapping SKOS, RDF, OWL and topic maps with upper ontologies has come up repeatedly, but with varying outcome. We value these articles as very important for our work, because of their excellent research and the experience of the authors.

- Pepper and Schwab [14] try to map the identification approach of XML Topic Maps to RDF, leaving a few issues open.
- Jack Park and Adam Cheyer mapped Topic Maps to Semantic Desktops for Personal Information Management in [13].
- The *Semex* system provides ideas about reference reconciliation [6]
- Jerome Euzenat proposed a top-level ontology for PIM in light of FOAF[32] Although this is a small, seemingly unimportant footnote, it shows how often capable people tried to address this problem.
- User Profile Ontology version 1[8], mentioned in [4][33].
- Latif and Tjoa [11] map a user ontology against other top-level ontologies such as SUMO and DOLCE and use the LATCH approach from Richard Saul Wurman.

[32] http://www.w3.org/2001/sw/Europe/200210/calendar/SyncLink.html
[33] http://oceanis.mm.di.uoa.gr/pened/?category=publications

References

[1] ISO/IEC 13250, Topic Maps, second edition. http://www.y12.doe.gov/sgml/sc34/document/0322_files/iso13250-2nd-ed-v2.pdf, 19 May 2002.

[2] D. Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C recommendation. http://www.w3.org/TR/rdf-schema/, 10 February 2004.

[3] K. Votis C. Alexakos, B. Vassiliadis and S. Likothanassis. *A Multilayer Ontology Scheme for Integrated Searching in Distributed Hypermedia*, volume Adaptive and Personalized Semantic Web of *Studies in Computational Intelligence*. Springer, 2006.

[4] T. Catarci, A. Dix, A. Katifori, G. Lepouras, and A. Poggi. Task-centered information management. In C. Thanos and F. Borri, editors, *DELOS Conference 2007 on Working Notes*, pages 253–263, Tirrenia, Pisa, 13-14 February 2007.

[5] Andreas R. Dengel. Six thousand words about multi-perspective personal document management. In *Proc. EDM IEEE Workshop*. IEEE, Oct 2006.

[6] Xin Dong and Alon Y. Halevy. A platform for personal information management and integration. In *Proc. of the CIDR Conference*, pages 119–130, 2005.

[7] Alistair Miles (edt). Simple knowledge organisation system (skos). http://www.w3.org/2004/02/skos/, Feb 2004.

[8] M. Golemati, A. Katifori, C. Vassilakis, G. Lepouras, and C. Halatsis. User profile ontology version 1. http://oceanis.mm.di.uoa.gr/pened/?category=publications, 2006.

[9] Harald Holz, Heiko Maus, Ansgar Bernardi, and Oleg Rostanin. From Lightweight, Proactive Information Delivery to Business Process-Oriented Knowledge Management. volume 0, pages 101–127, 2005.

[10] William Jones and Jamie Teevan, editors. *Personal Information Management*. University of Washington Press, October 2007.

[11] Khalid Latif and A Min Tjoa. Combining context ontology and landmarks for personal information management. In *Proceedings of International Conference on Computing and Informatics (ICOCI)*, Kuala Lumpur, Malaysia, June 2006.

[12] Alistair Miles, Thomas Baker, and Ralph Swick. Best practice recipes for publishing RDF vocabularies. W3C working draft, W3C, Mar 2006. http://www.w3.org/TR/swbp-vocab-pub/.

[13] Jack Park and Adam Cheyer. Just for me: Topic maps and ontologies. In Lutz Maicher and Jack Park, editors, *TMRA Charting the Topic Maps Research and Applications Landscape, First International Workshop on Topic Maps Research and Applications*, volume 3873 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2005.

[14] Steve Pepper and Sylvia Schwab. Curing the web's identity crisis. subject indicators for rdf. Technical report, Ontopia, 2003.

[15] Holger Rath. The topic maps handbook — detailed description of the standard and practical guidelines for using it in knowledge management. empolis white paper, empolis GmbH, 2003.

[16] Jean Rohmer. Lessons for the future of semantic desktops learnt from 10 years of experience with the ideliance semantic networks manager. In Stefan Decker, Jack Park, Dennis Quan, and Leo Sauermann, editors, *Proc. of Semantic Desktop Workshop at the ISWC, Galway, Ireland, November 6*, volume 175, November 2005.

[17] Leo Sauermann, Ludger van Elst, and Andreas Dengel. Pimo - a framework for representing personal information models. In Tassilo Pellegrini and Sebastian Schaffert, editors, *Proceedings of I-Semantics' 07*, pages pp. 270–277. JUCS, 2007.

[18] Graham Moore (eds). Steve Pepper. XML Topic Maps (XTM) 1.0. Specification, TopicMaps.Org, 2001.

A PIMO Specification

A.1 Ontology Classes Description

A.1.1 Agent

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	*pimo:*Organization A.1.20 p. 49 *pimo:*Person A.1.22 p. 50 *pimo:*PersonGroup A.1.23 p. 50
In domain of:	*pimo:*createdPimo A.2.9 p. 57 *pimo:*isOrganizationMemberOf A.2.28 p. 63
In range of:	*pimo:*creator A.2.10 p. 57 *pimo:*hasOrganizationMember A.2.20 p. 60
Description	An agent (eg. person, group, software or physical artifact). The Agent class is the class of agents; things that do stuff. A well known sub-class is Person, representing people. Other kinds of agents include Organization and Group. (inspired by FOAF). Agent is not a subclass of NAO:Party.

A.1.2 Association

Superclasses	*pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource
Subclasses	*pimo:*Attendee A.1.3 p. 42 *pimo:*OrganizationMember A.1.21 p. 50 *pimo:*PersonRole A.1.24 p. 51
In domain of:	*pimo:*associationEffectual A.2.1 p. 55 *pimo:*associationMember A.2.2 p. 55
In range of:	–
Description	An association between two or more pimo-things. This is used to model n-ary relations and metadata about relations. For example, the asociation of a person being organizational member is only effectual within a period of time (after the person joined the organization and before the person left the organization). There can be multiple periods of time when associations are valid.

A.1.3 Attendee

Superclasses	*pimo:*Association A.1.2 p. 42 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*PersonRole A.1.24 p. 51 *rdfs:*Resource

Subclasses	–
In domain of:	*pimo:*attendingMeeting A.2.4 p. 56
In range of:	–
Description	The role of someone attending a social event.

A.1.4 BlogPost

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Document A.1.13 p. 47 *pimo:*LogicalMediaType A.1.17 p. 48 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–
Description	A blog note. You just want to write something down right now and need a place to do that. Add a blog-note! This is an example class for a document type, there are more detailled ontologies to model Blog-Posts (like SIOC).

A.1.5 Building

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Location A.1.16 p. 48 *rdfs:*Resource *geo:*SpatialThing *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–
Description	A structure that has a roof and walls and stands more or less permanently in one place; "there was a three-story building on the corner"; "it was an imposing edifice". (Definition from SUMO).

A.1.6 City

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Location A.1.16 p. 48 *rdfs:*Resource *geo:*SpatialThing *pimo:*Thing A.1.32 p. 54
Subclasses	–

In domain of:	–
In range of:	–
Description	A large and densely populated urban area; may include several independent administrative districts; "Ancient Troy was a great city". (Definition from SUMO)

A.1.7 ClassOrThing

Superclasses	*pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource
Subclasses	*pimo:*Agent A.1.1 p. 42 *pimo:*BlogPost A.1.4 p. 43 *pimo:*Building A.1.5 p. 43 *pimo:*City A.1.6 p. 43 *pimo:*Collection A.1.10 p. 45 *pimo:*Contract A.1.11 p. 46 *pimo:*Country A.1.12 p. 46 *pimo:*Document A.1.13 p. 47 *pimo:*Event A.1.14 p. 47 *pimo:*Locatable A.1.15 p. 47 *pimo:*Location A.1.16 p. 48 *pimo:*LogicalMediaType A.1.17 p. 48 *pimo:*Meeting A.1.18 p. 49 *pimo:*Note A.1.19 p. 49 *pimo:*Organization A.1.20 p. 49 *pimo:*Person A.1.22 p. 50 *pimo:*PersonGroup A.1.23 p. 50 *pimo:*ProcessConcept A.1.26 p. 52 *pimo:*Project A.1.27 p. 52 *pimo:*Room A.1.28 p. 52 *pimo:*SocialEvent A.1.29 p. 53 *pimo:*State A.1.30 p. 53 *pimo:*Task A.1.31 p. 53 *pimo:*Thing A.1.32 p. 54 *pimo:*Topic A.1.33 p. 55
In domain of:	*pimo:*hasFolder A.2.18 p. 60 *pimo:*wikiText A.2.45 p. 67
In range of:	–
Description	Superclass of class and thing. To add properties to both class and thing.

A.1.8 ClassOrThingOrPropertyOrAssociation

Superclasses	*rdfs:*Resource

Subclasses	*pimo:*Agent A.1.1 p. 42 *pimo:*Association A.1.2 p. 42 *pimo:*Attendee A.1.3 p. 42 *pimo:*BlogPost A.1.4 p. 43 *pimo:*Building A.1.5 p. 43 *pimo:*City A.1.6 p. 43 *pimo:*ClassOrThing A.1.7 p. 44 *pimo:*Collection A.1.10 p. 45 *pimo:*Contract A.1.11 p. 46 *pimo:*Country A.1.12 p. 46 *pimo:*Document A.1.13 p. 47 *pimo:*Event A.1.14 p. 47 *pimo:*Locatable A.1.15 p. 47 *pimo:*Location A.1.16 p. 48 *pimo:*LogicalMediaType A.1.17 p. 48 *pimo:*Meeting A.1.18 p. 49 *pimo:*Note A.1.19 p. 49 *pimo:*Organization A.1.20 p. 49 *pimo:*OrganizationMember A.1.21 p. 50 *pimo:*Person A.1.22 p. 50 *pimo:*PersonGroup A.1.23 p. 50 *pimo:*PersonRole A.1.24 p. 51 *pimo:*ProcessConcept A.1.26 p. 52 *pimo:*Project A.1.27 p. 52 *pimo:*Room A.1.28 p. 52 *pimo:*SocialEvent A.1.29 p. 53 *pimo:*State A.1.30 p. 53 *pimo:*Task A.1.31 p. 53 *pimo:*Thing A.1.32 p. 54 *pimo:*Topic A.1.33 p. 55
In domain of:	*pimo:*isDefinedBy A.2.26 p. 62
In range of:	–
Description	Superclass of resources that can be generated by the user.

A.1.9 ClassRole

Superclasses	*rdfs:*Resource
Subclasses	–
In domain of:	–
In range of:	*pimo:*classRole A.2.7 p. 57
Description	Roles of classes in PIMO: concrete instances are Abstract and Concrete.
Instances	*pimo:*AbstractClass *pimo:*ConcreteClass

A.1.10 Collection

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54

Subclasses	*pimo:*PersonGroup A.1.23 p. 50
In domain of:	–
In range of:	–
Description	A collection of Things, independent of their class. The items in the collection share a common property. Which property may be modelled explicitly or mentioned in the description of the Collection. The requirement of explicit modelling the semantic meaning of the collection is not mandatory, as collections can be created ad-hoc. Implizit modelling can be applied by the system by learning the properties. For example, a Collection of "Coworkers" could be defined as that all elements must be of class "Person" and have an attribute "work for the same Organization as the user". Further standards can be used to model these attributes.

A.1.11 Contract

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Document A.1.13 p. 47 *pimo:*LogicalMediaType A.1.17 p. 48 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–
Description	A binding agreement between two or more persons that is enforceable by law. (Definition from SUMO). This is an example class for a document type, there are more detailled ontologies to model Contracts.

A.1.12 Country

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Location A.1.16 p. 48 *rdfs:*Resource *geo:*SpatialThing *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–
Description	The territory occupied by a nation; "he returned to the land of his birth"; "he visited several European countries". (Definition from SUMO)

A.1.13 Document

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*LogicalMediaType A.1.17 p. 48 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	*pimo:*BlogPost A.1.4 p. 43 *pimo:*Contract A.1.11 p. 46 *pimo:*Note A.1.19 p. 49
In domain of:	–
In range of:	–
Description	A generic document. This is a placeholder class for document-management domain ontologies to subclass. Create more and specified subclasses of pimo:Document for the document types in your domain. Documents are typically instances of both NFO:Document (modeling the information element used to store the document) and a LogicalMediaType subclass. Two examples are given for what to model here: a contract for a business domain, a BlogPost for an informal domain.

A.1.14 Event

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*ProcessConcept A.1.26 p. 52 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	*pimo:*Meeting A.1.18 p. 49 *pimo:*SocialEvent A.1.29 p. 53
In domain of:	–
In range of:	–
Description	Something that happens An Event is conceived as compact in time. (Definition from Merriam-Webster)

A.1.15 Locatable

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	*pimo:*Meeting A.1.18 p. 49 *pimo:*Organization A.1.20 p. 49 *pimo:*Person A.1.22 p. 50 *pimo:*SocialEvent A.1.29 p. 53
In domain of:	*pimo:*hasLocation A.2.19 p. 60

In range of:	–
Description	Things that can be at a location. Abstract class, use it as a superclass of things that can be placed in physical space.

A.1.16 Location

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource *geo:*SpatialThing *pimo:*Thing A.1.32 p. 54
Subclasses	*pimo:*Building A.1.5 p. 43 *pimo:*City A.1.6 p. 43 *pimo:*Country A.1.12 p. 46 *pimo:*Room A.1.28 p. 52 *pimo:*State A.1.30 p. 53
In domain of:	*pimo:*containsLocation A.2.8 p. 57 *pimo:*isLocationOf A.2.27 p. 62 *pimo:*locatedWithin A.2.33 p. 64
In range of:	*pimo:*containsLocation A.2.8 p. 57 *pimo:*hasLocation A.2.19 p. 60 *pimo:*locatedWithin A.2.33 p. 64
Description	A physical location. Subclasses are modeled for the most common locations humans work in: Building, City, Country, Room, State. This selection is intended to be applicable cross-cultural and cross-domain. City is a prototype that can be further refined for villages, etc. Subclass of a WGS84:SpatialThing, can have geo-coordinates.

A.1.17 LogicalMediaType

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	*pimo:*BlogPost A.1.4 p. 43 *pimo:*Contract A.1.11 p. 46 *pimo:*Document A.1.13 p. 47 *pimo:*Note A.1.19 p. 49
In domain of:	–
In range of:	–

Description	Logical media types represent the content aspect of information elements e.g. a flyer, a contract, a promotional video, a todo list. The user can create new logical media types dependend on their domain: a salesman will need MarketingFlyer, Offer, Invoice while a student might create Report, Thesis and Homework. This is independent from the information element and data object (NIE/NFO) in which the media type will be stored. The same contract can be stored in a PDF file, a text file, or an HTML website. The groundingOccurrence of a LogicalMediaType is the Document that stores the content.

A.1.18 Meeting

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Event A.1.14 p. 47 *pimo:*Locatable A.1.15 p. 47 *pimo:*ProcessConcept A.1.26 p. 52 *rdfs:*Resource *pimo:*SocialEvent A.1.29 p. 53 *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–
Description	The social act of assembling for some common purpose; "his meeting with the salesman was the high point of his day". (Definition from SUMO)

A.1.19 Note

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Document A.1.13 p. 47 *pimo:*LogicalMediaType A.1.17 p. 48 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–
Description	A note. The textual contents of the note should be expressed in the nao:description value of the note.

A.1.20 Organization

Superclasses	*pimo:*Agent A.1.1 p. 42 *pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Locatable A.1.15 p. 47 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	*pimo:*hasOrganizationMember A.2.20 p. 60
In range of:	*pimo:*isOrganizationMemberOf A.2.28 p. 63 *pimo:*organization A.2.37 p. 65
Description	An administrative and functional structure (as a business or a political party). (Definition from Merriam-Webster)

A.1.21 OrganizationMember

Superclasses	*pimo:*Association A.1.2 p. 42 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*PersonRole A.1.24 p. 51 *rdfs:*Resource
Subclasses	–
In domain of:	*pimo:*organization A.2.37 p. 65
In range of:	–
Description	The role of one or multiple persons being a member in one or multiple organizations. Use pimo:organization and pimo:roleHolder to link to the organizations and persons.

A.1.22 Person

Superclasses	*pimo:*Agent A.1.1 p. 42 *pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Locatable A.1.15 p. 47 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	*pimo:*attends A.2.5 p. 56 *pimo:*jabberId A.2.32 p. 64
In range of:	*pimo:*attendee A.2.3 p. 56 *pimo:*roleHolder A.2.41 p. 66
Description	Represents a person. Either living, dead, real or imaginary. (Definition from foaf:Person)

A.1.23 PersonGroup

Superclasses	*pimo:*Agent A.1.1 p. 42 *pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Collection A.1.10 p. 45 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–
Description	A group of Persons. They are connected to each other by sharing a common attribute, for example they all belong to the same organization or have a common interest. Refer to pimo:Collection for more information about defining collections.

A.1.24 PersonRole

Superclasses	*pimo:*Association A.1.2 p. 42 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource
Subclasses	*pimo:*Attendee A.1.3 p. 42 *pimo:*OrganizationMember A.1.21 p. 50
In domain of:	*pimo:*roleContext A.2.40 p. 66 *pimo:*roleHolder A.2.41 p. 66
In range of:	–
Description	A person takes a certain role in a given context. The role can be that of "a mentor or another person" or "giving a talk at a meeting", etc.

A.1.25 PersonalInformationModel

Superclasses	*nrl:*Data *nrl:*Graph *nrl:*InstanceBase *nrl:*KnowledgeBase *nrl:*Ontology *rdfs:*Resource *nrl:*Schema
Subclasses	–
In domain of:	*pimo:*creator A.2.10 p. 57
In range of:	*pimo:*createdPimo A.2.9 p. 57 *pimo:*isDefinedBy A.2.26 p. 62
Description	A Personal Information Model (PIMO) of a user. Represents the sum of all information from the personal knowledge workspace (in literature also referred to as Personal Space of Information (PSI)) which a user needs for Personal Information Management (PIM).

A.1.26 ProcessConcept

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	*pimo:*Event A.1.14 p. 47 *pimo:*Meeting A.1.18 p. 49 *pimo:*Project A.1.27 p. 52 *pimo:*SocialEvent A.1.29 p. 53 *pimo:*Task A.1.31 p. 53
In domain of:	*pimo:*dtend A.2.12 p. 58 *pimo:*dtstart A.2.13 p. 58
In range of:	–
Description	Concepts that relate to a series of actions or operations conducing to an end. Abstract class. Defines optional start and endtime properties, names taken from NCAL.

A.1.27 Project

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*ProcessConcept A.1.26 p. 52 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–
Description	Any piece of work that is undertaken or attempted (Wordnet). An enterprise carefully planned to achieve a particular aim (Oxford Dictionary).

A.1.28 Room

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Location A.1.16 p. 48 *rdfs:*Resource *geo:*SpatialThing *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–

Description	A properPart of a Building which is separated from the exterior of the Building and/or other Rooms of the Building by walls. Some Rooms may have a specific purpose, e.g. sleeping, bathing, cooking, entertainment, etc. (Definition from SUMO).

A.1.29 SocialEvent

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Event A.1.14 p. 47 *pimo:*Locatable A.1.15 p. 47 *pimo:*ProcessConcept A.1.26 p. 52 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	*pimo:*Meeting A.1.18 p. 49
In domain of:	*pimo:*attendee A.2.3 p. 56 *pimo:*duration A.2.14 p. 58
In range of:	*pimo:*attendingMeeting A.2.4 p. 56 *pimo:*attends A.2.5 p. 56
Description	A social occasion or activity. (Definition from Merriam-Webster)

A.1.30 State

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*Location A.1.16 p. 48 *rdfs:*Resource *geo:*SpatialThing *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	–
In range of:	–
Description	Administrative subdivisions of a Nation that are broader than any other political subdivisions that may exist. This Class includes the states of the United States, as well as the provinces of Canada and European countries. (Definition from SUMO).

A.1.31 Task

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *pimo:*ProcessConcept A.1.26 p. 52 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	*pimo:*taskDueTime A.2.44 p. 67
In range of:	–
Description	A (usually assigned) piece of work (often to be finished within a certain time). (Definition from Merriam-Webster)

A.1.32 Thing

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource
Subclasses	*pimo:*Agent A.1.1 p. 42 *pimo:*BlogPost A.1.4 p. 43 *pimo:*Building A.1.5 p. 43 *pimo:*City A.1.6 p. 43 *pimo:*Collection A.1.10 p. 45 *pimo:*Contract A.1.11 p. 46 *pimo:*Country A.1.12 p. 46 *pimo:*Document A.1.13 p. 47 *pimo:*Event A.1.14 p. 47 *pimo:*Locatable A.1.15 p. 47 *pimo:*Location A.1.16 p. 48 *pimo:*LogicalMediaType A.1.17 p. 48 *pimo:*Meeting A.1.18 p. 49 *pimo:*Note A.1.19 p. 49 *pimo:*Organization A.1.20 p. 49 *pimo:*Person A.1.22 p. 50 *pimo:*PersonGroup A.1.23 p. 50 *pimo:*ProcessConcept A.1.26 p. 52 *pimo:*Project A.1.27 p. 52 *pimo:*Room A.1.28 p. 52 *pimo:*SocialEvent A.1.29 p. 53 *pimo:*State A.1.30 p. 53 *pimo:*Task A.1.31 p. 53 *pimo:*Topic A.1.33 p. 55
In domain of:	*pimo:*datatypeProperty A.2.11 p. 58 *pimo:*groundingOccurrence A.2.16 p. 59 *pimo:*hasDeprecatedRepresentation A.2.17 p. 59 *pimo:*hasOtherRepresentation A.2.22 p. 61 *pimo:*hasPart A.2.24 p. 61 *pimo:*hasTopic A.2.25 p. 62 *pimo:*isRelated A.2.29 p. 63 *pimo:*isTopicOf A.2.30 p. 63 *pimo:*objectProperty A.2.35 p. 64 *pimo:*occurrence A.2.36 p. 65 *pimo:*partOf A.2.38 p. 65 *pimo:*referencingOccurrence A.2.39 p. 66

In range of:	*pimo:*associationMember A.2.2 p. 55 *pimo:*hasPart A.2.24 p. 61 *pimo:*hasTopic A.2.25 p. 62 *pimo:*isLocationOf A.2.27 p. 62 *pimo:*isRelated A.2.29 p. 63 *pimo:*isTopicOf A.2.30 p. 63 *pimo:*objectProperty A.2.35 p. 64 *pimo:*partOf A.2.38 p. 65 *pimo:*roleContext A.2.40 p. 66
Description	Entities that are in the direct attention of the user when doing knowledge work.

A.1.33 Topic

Superclasses	*pimo:*ClassOrThing A.1.7 p. 44 *pimo:*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44 *rdfs:*Resource *pimo:*Thing A.1.32 p. 54
Subclasses	–
In domain of:	*pimo:*subTopic A.2.42 p. 66 *pimo:*superTopic A.2.43 p. 67
In range of:	*pimo:*subTopic A.2.42 p. 66 *pimo:*superTopic A.2.43 p. 67
Description	A topic is the subject of a discussion or document. Topics are distinguished from Things in their taxonomic nature, examples are scientific areas such as "Information Science", "Biology", or categories used in content syndication such as "Sports", "Politics". They are specific to the user's domain.

A.2 Ontology Properties Description

A.2.1 associationEffectual

Domain	*pimo*Association A.1.2 p. 42
Range	*rdfs*Resource
Superproperties	–
Subproperties	–
Description	During which time is this association effective? If omitted, the association is always effective. Start time and end-time may be left open, an open start time indicates that the fact is unknown, an open end-time indicates that the end-date is either unknown or the association has not ended. There can be multiple effectual periods.

A.2.2 associationMember

Domain	*pimo*Association A.1.2 p. 42

Range	*pimo*Thing A.1.32 p. 54
Superproperties	–
Subproperties	*pimo:*attendingMeeting A.2.4 p. 56 *pimo:*organization A.2.37 p. 65 *pimo:*roleContext A.2.40 p. 66 *pimo:*roleHolder A.2.41 p. 66
Description	An super-property of all roles that an entity can have in an association. Member is the generic role of a thing in an association. Association subclasses should define sub-properties of this property. Associations can have Things as

A.2.3 attendee

Domain	*pimo*SocialEvent A.1.29 p. 53
Range	*pimo*Person A.1.22 p. 50
Superproperties	*nao:*annotation *nao:*isRelated *pimo:*isRelated A.2.29 p. 63 *pimo:*objectProperty A.2.35 p. 64
Subproperties	–
Description	A social event is attended by a person.

A.2.4 attendingMeeting

Domain	*pimo*Attendee A.1.3 p. 42
Range	*pimo*SocialEvent A.1.29 p. 53
Superproperties	*pimo:*associationMember A.2.2 p. 55 *pimo:*roleContext A.2.40 p. 66
Subproperties	–
Description	the attended meeting

A.2.5 attends

Domain	*pimo*Person A.1.22 p. 50
Range	*pimo*SocialEvent A.1.29 p. 53
Superproperties	*nao:*annotation *nao:*isRelated *pimo:*isRelated A.2.29 p. 63 *pimo:*objectProperty A.2.35 p. 64
Subproperties	–
Description	A person attends a social event.

A.2.6 broader

Domain	
Range	
Superproperties	–

Subproperties	–
Description	

A.2.7 classRole

Domain	
Range	*pimo*ClassRole A.1.9 p. 45
Superproperties	–
Subproperties	–
Description	Annotating abstract and concrete classes. Implementations may offer the feature to hide abstract classes. By default, classes are concrete. Classes can be declared abstract by setting their classRole to abstract. Instances should not have an abstract class as type (if not inferred).

A.2.8 containsLocation

Domain	*pimo*Location A.1.16 p. 48
Range	*pimo*Location A.1.16 p. 48
Superproperties	*pimo:*hasPart A.2.24 p. 61 *pimo:*objectProperty A.2.35 p. 64
Subproperties	–
Description	The subject location contains the object location. For example, a building contains a room or a country contains a city.

A.2.9 createdPimo

Domain	*pimo*Agent A.1.1 p. 42
Range	*pimo*PersonalInformationModel A.1.25 p. 51
Superproperties	–
Subproperties	–
Description	The creator of the Personal Information Model. The human being whose mental models are represented in the PIMO.

A.2.10 creator

Domain	*pimo*PersonalInformationModel A.1.25 p. 51
Range	*pimo*Agent A.1.1 p. 42
Superproperties	*nao:*annotation *x:*creator *nao:*creator
Subproperties	–

Description	The creator of the Personal Information Model. A subproperty of NAO:creator. The human being whose mental models are represented in the PIMO. Range is an Agent.

A.2.11 datatypeProperty

Domain	*pimo*Thing A.1.32 p. 54
Range	
Superproperties	–
Subproperties	*geo:*alt *pimo:*dtend A.2.12 p. 58 *pimo:*dtstart A.2.13 p. 58 *pimo:*duration A.2.14 p. 58 *geo:*lat *geo:*long *pimo:*taskDueTime A.2.44 p. 67
Description	The object of statements is a literal, resource, or datatype value describing the subject thing. Users should be able to edit statements defined with this property. Abstract super-property.

A.2.12 dtend

Domain	*pimo*ProcessConcept A.1.26 p. 52
Range	*xsd*dateTime
Superproperties	*pimo:*datatypeProperty A.2.11 p. 58
Subproperties	–
Description	This property specifies the date and time when a process ends. Inspired by NCAL:dtend.

A.2.13 dtstart

Domain	*pimo*ProcessConcept A.1.26 p. 52
Range	*xsd*dateTime
Superproperties	*pimo:*datatypeProperty A.2.11 p. 58
Subproperties	–
Description	This property specifies when the process begins. Inspired by NCAL:dtstart.

A.2.14 duration

Domain	*pimo*SocialEvent A.1.29 p. 53
Range	*rdfs*Resource
Superproperties	*pimo:*datatypeProperty A.2.11 p. 58
Subproperties	–
Description	The duration of the meeting. Begin and end time.

A.2.15 groundingForDeletedThing

Domain	
Range	*rdfs*Resource
Superproperties	–
Subproperties	–
Description	This NIE Information Element was used as a grounding occurrence for the object Thing. The Thing was then deleted by the user manually, indicating that this Information Element should not cause an automatic creation of another Thing in the future. The object resource has no range to indicate that it was completely removed from the user's PIMO, including the rdf:type statement. Relevant for data alignment and enrichment algorithms.

A.2.16 groundingOccurrence

Domain	*pimo*Thing A.1.32 p. 54
Range	*nie*InformationElement **??** p. **??**
Superproperties	*pimo*:occurrence A.2.36 p. 65
Subproperties	–
Description	The subject Thing represents the entity that is described in the object InformationElement. The subject Thing is the canonical, unique representation in the personal information model for the entity described in the object. Multiple InformationElements can be the grounding occurrence of the same Thing, one InformationElement can be the groundingOccurrence of only one Thing.

A.2.17 hasDeprecatedRepresentation

Domain	*pimo*Thing A.1.32 p. 54
Range	*rdfs*Resource
Superproperties	–
Subproperties	–
Description	The subject Thing was represented previously using the object resource. This indicates that the object resource was a duplicate representation of the subject and merged with the subject. Implementations can use this property to resolve dangling links in distributed system. When encountering resources that are deprecated representations of a Thing, they should be replaced with the Thing. The range is not declared as we assume all knowledge about the object is gone, including its rdf:type.

A.2.18 hasFolder

Domain	*pimo*ClassOrThing A.1.7 p. 44
Range	*nfo*Folder **??** p. **??**
Superproperties	–
Subproperties	–
Description	Folders can be used to store information elements related to a Thing or Class. This property can be used to connect a Class or Thing to existing Folders. Implementations can suggest annotations for documents stored inside these folders or suggest the folder for new documents related to the Thing or Class.

A.2.19 hasLocation

Domain	*pimo*Locatable A.1.15 p. 47
Range	*pimo*Location A.1.16 p. 48
Superproperties	*nao:*annotation *nao:*isRelated *pimo:*isRelated A.2.29 p. 63 *pimo:*objectProperty A.2.35 p. 64
Subproperties	–
Description	The subject thing is currently located at the object location.

A.2.20 hasOrganizationMember

Domain	*pimo*Organization A.1.20 p. 49
Range	*pimo*Agent A.1.1 p. 42
Superproperties	*pimo:*hasPart A.2.24 p. 61 *pimo:*objectProperty A.2.35 p. 64
Subproperties	–
Description	The subject organization has the object person or organization (Agent) as a member.

A.2.21 hasOtherConceptualization

Domain	*rdfs*Class
Range	*rdfs*Class
Superproperties	*pimo:*occurrence A.2.36 p. 65 *rdfs:*subClassOf
Subproperties	–

Description	Short: hasOtherRepresentation points from a Class in your PIMO to a class in a domain ontology that represents the same class. Longer: hasOtherConceptualization means that a class of real world objects O represented by a concept C1 in the ontology has additional conceptualizations (as classes C2-Cn in different domain ontologies). This means: IF (O_i is conceptialized by C_j in Ontology_k) AND (O_l is conceptialized by C_m in Ontology_n) THEN (O_i and O_l is the same set of objects). hasOtherConceptualization is an transitive relation, but not equivalent (not symmetric nor reflexive).

A.2.22 hasOtherRepresentation

Domain	*pimo*Thing A.1.32 p. 54
Range	*rdfs*Resource
Superproperties	*pimo:*occurrence A.2.36 p. 65
Subproperties	–
Description	hasOtherRepresentation points from a Thing in your PIMO to a thing in an ontology that represents the same real world thing. This means that the real world object O represented by an instance I1 has additional representations (as instances I2-In of different conceptualizations). This means: IF (I_i represents O_j in Ontology_k) AND (I_m represents O_n in Ontology_o) THEN (O_n and O_j are the same object). hasOtherRepresentation is a transitive relation, but not equivalent (not symmetric nor reflexive). For example, the URI of a foaf:Person representation published on the web is a hasOtherRepresentation for the person. This property is inverse functional, two Things from two information models having the same hasOtherRepresentation are considered to be representations of the same entity from the real world. TODO: rename this to subjectIndicatorRef to resemble topic maps ideas?

A.2.23 hasOtherSlot

Domain	*rdf*Property
Range	*rdf*Property
Superproperties	*rdfs:*subPropertyOf
Subproperties	–
Description	hasOtherSlot points from a clot in your PIMO to a slot in a domain ontology that represents the same connection idea.

A.2.24 hasPart

Domain	*pimo*Thing A.1.32 p. 54
Range	*pimo*Thing A.1.32 p. 54
Superproperties	*pimo:*objectProperty A.2.35 p. 64
Subproperties	*pimo:*containsLocation A.2.8 p. 57 *pimo:*hasOrganizationMember A.2.20 p. 60 *pimo:*subTopic A.2.42 p. 66
Description	The object is part of the subject. Like a page is part of a book or an engine is part of a car. You can make sub-properties of this to reflect more detailed relations.

A.2.25 hasTopic

Domain	*pimo*Thing A.1.32 p. 54
Range	*pimo*Thing A.1.32 p. 54
Superproperties	*nao:*annotation *nao:*hasTopic *nao:*isRelated *pimo:*objectProperty A.2.35 p. 64
Subproperties	–
Description	The subject's contents describes the object. Or the subject can be seen as belonging to the topic described by the object. Similar semantics as skos:subject.

A.2.26 isDefinedBy

Domain	*pimo*ClassOrThingOrPropertyOrAssociation A.1.8 p. 44
Range	*pimo*PersonalInformationModel A.1.25 p. 51
Superproperties	–
Subproperties	–
Description	Each element in a PIMO must be connected to the PIMO, to be able to track multiple PIMOs in a distributed scenario. Also, this is the way to find the user that this Thing belongs to.

A.2.27 isLocationOf

Domain	*pimo*Location A.1.16 p. 48
Range	*pimo*Thing A.1.32 p. 54
Superproperties	*nao:*annotation *nao:*isRelated *pimo:*isRelated A.2.29 p. 63 *pimo:*objectProperty A.2.35 p. 64
Subproperties	–
Description	The subject location is the current location of the object.

A.2.28 isOrganizationMemberOf

Domain	*pimo*Agent A.1.1 p. 42
Range	*pimo*Organization A.1.20 p. 49
Superproperties	*pimo:*objectProperty A.2.35 p. 64 *pimo:*partOf A.2.38 p. 65
Subproperties	–
Description	The subject person or organozation (Agent) is member of the object organization.

A.2.29 isRelated

Domain	*pimo*Thing A.1.32 p. 54
Range	*pimo*Thing A.1.32 p. 54
Superproperties	*nao:*annotation *nao:*isRelated *pimo:*objectProperty A.2.35 p. 64
Subproperties	*pimo:*attendee A.2.3 p. 56 *pimo:*attends A.2.5 p. 56 *pimo:*hasLocation A.2.19 p. 60 *pimo:*isLocationOf A.2.27 p. 62
Description	The thing is related to the other thing. Similar in meaning to skos:related. Symmetric but not transitive.

A.2.30 isTopicOf

Domain	*pimo*Thing A.1.32 p. 54
Range	*pimo*Thing A.1.32 p. 54
Superproperties	*nao:*annotation *nao:*isRelated *nao:*isTopicOf *pimo:*objectProperty A.2.35 p. 64
Subproperties	–
Description	This thing is described further in the object thing. Similar semantics as skos:isSubjectOf.

A.2.31 isWriteable

Domain	
Range	*rdfs*Literal
Superproperties	–
Subproperties	–
Description	Defines if this information model can be modified by the user of the system. This is usually false for imported ontologies and true for the user's own PersonalInformationModel.

A.2.32 jabberId

Domain	*pimo*Person A.1.22 p. 50
Range	*rdfs*Literal
Superproperties	–
Subproperties	–
Description	Jabber-ID of the user. Used to communicate amongst peers in the social scenario of the semantic desktop. Use the xmpp node identifier as specified by RFC3920, see http://www.xmpp.org/specs/rfc3920.html#addressing-node. The format is the same as e-mail addresses: username@hostname.

A.2.33 locatedWithin

Domain	*pimo*Location A.1.16 p. 48
Range	*pimo*Location A.1.16 p. 48
Superproperties	*pimo*:objectProperty A.2.35 p. 64 *pimo*:partOf A.2.38 p. 65
Subproperties	–
Description	The subject location is contained within the object location. For example, a room is located within a building or a city is located within a country.

A.2.34 narrower

Domain	
Range	
Superproperties	–
Subproperties	–
Description	

A.2.35 objectProperty

Domain	*pimo*Thing A.1.32 p. 54
Range	*pimo*Thing A.1.32 p. 54
Superproperties	–

Subproperties	*pimo*:attendee A.2.3 p. 56 *pimo*:attends A.2.5 p. 56 *pimo*:containsLocation A.2.8 p. 57 *pimo*:hasLocation A.2.19 p. 60 *pimo*:hasOrganizationMember A.2.20 p. 60 *pimo*:hasPart A.2.24 p. 61 *pimo*:hasTopic A.2.25 p. 62 *pimo*:isLocationOf A.2.27 p. 62 *pimo*:isOrganizationMemberOf A.2.28 p. 63 *pimo*:isRelated A.2.29 p. 63 *pimo*:isTopicOf A.2.30 p. 63 *pimo*:locatedWithin A.2.33 p. 64 *pimo*:partOf A.2.38 p. 65 *pimo*:subTopic A.2.42 p. 66 *pimo*:superTopic A.2.43 p. 67
Description	The object of statements is another Thing. Users should be able to edit statements defined with this property. Abstract super-property.

A.2.36 occurrence

Domain	*pimo*Thing A.1.32 p. 54
Range	*rdfs*Resource
Superproperties	–
Subproperties	*pimo*:groundingOccurrence A.2.16 p. 59 *pimo*:hasOtherConceptualization A.2.21 p. 60 *pimo*:hasOtherRepresentation A.2.22 p. 61
Description	The subject Thing is represented also in the object resource. All facts added to the object resource are valid for the subject thing. The subject is the canonical represtation of the object. In particual, this implies when (?object ?p ?v) -> (?subject ?p ?v) and (?s ?p ?object) -> (?s ?p ?subject). The class of the object is not defined, but should be compatible with the class of the subject. Occurrence relations can be inferred through same identifiers or referencingOccurrence relations.

A.2.37 organization

Domain	*pimo*OrganizationMember A.1.21 p. 50
Range	*pimo*Organization A.1.20 p. 49
Superproperties	*pimo*:associationMember A.2.2 p. 55
Subproperties	–
Description	relation to the organization in an Organization-Member association.

A.2.38 partOf

Domain	*pimo*Thing A.1.32 p. 54
Range	*pimo*Thing A.1.32 p. 54

Superproperties	*pimo:*objectProperty A.2.35 p. 64
Subproperties	*pimo:*isOrganizationMemberOf A.2.28 p. 63 *pimo:*locatedWithin A.2.33 p. 64 *pimo:*superTopic A.2.43 p. 67
Description	This is part of the object. Like a page is part of a book or an engine is part of a car. You can make sub-properties of this to reflect more detailed relations.

A.2.39 referencingOccurrence

Domain	*pimo*Thing A.1.32 p. 54
Range	*nie*InformationElement **??** p. **??**
Superproperties	–
Subproperties	–
Description	The subject thing is described in the object document. Ideally, the document is public and its primary topic is the thing. Although this property is not inverse-functional (because the Occurrences are not canonical elements of a formal ontology) this property allows to use public documents, such as wikipedia pages, as indicators identity. The more formal hasOtherRepresentation property can be used when an ontology about the subject exists.

A.2.40 roleContext

Domain	*pimo*PersonRole A.1.24 p. 51
Range	*pimo*Thing A.1.32 p. 54
Superproperties	*pimo:*associationMember A.2.2 p. 55
Subproperties	*pimo:*attendingMeeting A.2.4 p. 56
Description	The context where the role-holder impersonates this role. For example, the company where a person is employed.

A.2.41 roleHolder

Domain	*pimo*PersonRole A.1.24 p. 51
Range	*pimo*Person A.1.22 p. 50
Superproperties	*pimo:*associationMember A.2.2 p. 55
Subproperties	–
Description	the person taking the role

A.2.42 subTopic

Domain	*pimo*Topic A.1.33 p. 55
Range	*pimo*Topic A.1.33 p. 55

Superproperties	*pimo*:hasPart A.2.24 p. 61 *pimo*:objectProperty A.2.35 p. 64
Subproperties	–
Description	The object topic is more specific in meaning than the subject topic. Transitive. Similar in meaning to skos:narrower

A.2.43 superTopic

Domain	*pimo*Topic A.1.33 p. 55
Range	*pimo*Topic A.1.33 p. 55
Superproperties	*pimo*:objectProperty A.2.35 p. 64 *pimo*:partOf A.2.38 p. 65
Subproperties	–
Description	The object topic is more general in meaning than the subject topic. Transitive. Similar to skos:broader.

A.2.44 taskDueTime

Domain	*pimo*Task A.1.31 p. 53
Range	*xsd*dateTime
Superproperties	*pimo*:datatypeProperty A.2.11 p. 58
Subproperties	–
Description	when is this task due? Represented in ISO 8601, example: 2003-11-22T17:00:00

A.2.45 wikiText

Domain	*pimo*ClassOrThing A.1.7 p. 44
Range	*rdfs*Literal
Superproperties	–
Subproperties	–
Description	A wiki-like free-text description of a Thing or a Class. The text can be formatted using a limited set of HTML elements and can contain links to other Things. The format is described in detail in the WIF specification (http://semanticweb.org/wiki/Wiki_Interchange_Format).