EU-IST Project IST-2003-506826 SEKT

SEKT: Semantically Enabled Knowledge Technologies

# D1.8.1 Base upper-level ontology (BULO) Guidance[1]

**Ivan Terziev**

**Atanas Kiryakov**

**Dimitar Manov**

(Ontotext Lab, Sirma Group – all the authors)

**Abstract**

An important practical approach to ontology generation is the use of background or pre-existing knowledge in the form of a basic upper-level ontology. Such an ontology can also be used for metadata generation and as a groundwork for the overall knowledge modelling and integration strategy of a KM environment.

The essential contribution of this deliverable is a basic upper-level ontology called **PROTON** (PROTo ONtology), which is hereby introduced and documented. It contains about 300 classes and 100 properties, providing coverage of the general concepts necessary for a wide range of tasks, including semantic annotation, indexing, and retrieval of documents. The design principles can be summarized as follows (i) domain-independence; (ii) light-weight logical definitions; (iii) alignment with popular standards; (iv) good coverage of named entities and concrete domains (i.e. people, organizations, locations, numbers, dates, addresses). The ontology is originally encoded in a fragment of OWL Lite and split into four modules: System, Top, Upper, and KM (Knowledge Management).

**Keyword list:** upper-level ontology, knowledge engineering, knowledge modelling

**WP1: Ontology Generation**

| | |
|---|---|
| Software | PU |
| | 12.07.2005 |

[1] For a number of reasons, the initial name of the ontology, BULO, was changed to PROTON, thus the actual name PROTON has been used instead of BULO throughout this document.

## SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

**British Telecommunications plc.**
Orion 5/12, Adastral Park
Ipswich IP5 3RE
UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

**Empolis GmbH**
Europaallee 10
67657 Kaiserslautern
Germany
Tel: +49 631 303 5540
Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

**Jozef Stefan Institute**
Jamova 39
1000 Ljubljana
Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

**University of Karlsruhe**, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 6592
Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

**University of Sheffield**
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1891
Fax: +44 114 222 1810
Contact person: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

**University of Innsbruck**
Institute of Computer Science
Techikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475
Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

**Intelligent Software Components S.A.**
Pedro de Valvidia, 10
28006
Madrid
Spain
Tel: +34 913 349 797
Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

**Kea-pro GmbH**
Tal
6464 Springen
Switzerland
Tel: +41 41 879 00
Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

**Ontoprise GmbH**
Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912
Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

**Sirma AI EAD, Ontotext Lab.**
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

**Vrije Universiteit Amsterdam (VUA)**
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

**Universitat Autonoma de Barcelona**
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vall` es)
Barcelona
Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovasquab.es

# 1 Executive Summary

An important practical approach to ontology generation is the use of background or pre-existing knowledge in the form of a basic upper-level ontology. Such an ontology can also be used for metadata generation (WP2) and as a groundwork for the overall knowledge modelling and integration strategy of a KM environment.

PROTON ontology contains about 300 classes and 100 properties, providing coverage of the general concepts necessary for a wide range of tasks, including semantic annotation, indexing, and retrieval of documents. The design principles can be summarized as follows (i) domain-independence; (ii) light-weight logical definitions; (iii) alignment with popular standards; (iv) good coverage of named entities and concrete domains (i.e. people, organizations, locations, numbers, dates, addresses).

The essential contribution of this deliverable is the PROTON ontology itself, which is hereby introduced and documented. The development of the ontology is organized in quarterly cycles (release, application, collecting feedback). Being derived from the KIMO ontology (developed for the early versions of the KIM platform), PROTON has been developed significantly further, as follows:

- represented in OWL Lite (versus RDFS for KIMO);
- broken down into modules (System, Top, Upper, Knowledge Management);
- KIM-independent (the KIMO-related concepts were relocated to specific, separate modules);
- some tuning took place in response to the requirements of SEKT project (WP5 “Knowledge Access” and the digital library case studies, WP11).

This guidance is complementary to the PROTON ontology, which provides its own internal descriptions (definitions) for most classes and properties, and on the whole the latter are likely to prove sufficient enough when one needs information about a specific class or property. However, this document is informative as it:

- presents an overall, cohesive view of PROTON;
- the rationales and decisions that the design ot PROTON is grounded on;
- provides useful comments on the alignment or PROTON to other ontologies and metadata schemata; and
- puts an introductory exposition of the essential layers and branches in PROTON on view.

Lastly, it is worth noting that the PROTON is constantly being developed. A community process is yet to be initiated (see section 10.1) so that an extensive discussion over PROTON can be carried out to guaranttee its openness and gradual improvement as a basis for different applications and various domains. The community process adapts the DILIGENT methodology (from WP7).

# 2 Contents

**1 Executive Summary ....................................................................................3**
**2 Contents .....................................................................................................4**
**3 Introduction...............................................................................................7**
3.1 How PROTON Relates to KIM and KIMO..................................................8
3.2 Scope, Coverage, Compliance ......................................................................9
3.3 Ontologies and Knowledge Representation..................................................9
3.4 PROTON: Glimpses of a Philosopher's Stance ..........................................10
3.5 Ontology Languages ....................................................................................12
3.6 Physical Address and Namespaces ..............................................................13
**4 Design Rationales .....................................................................................14**
4.1 Ontologies as RDBMS Schema ...................................................................14
4.2 Formalization (Knowledge Representation) Approach ...............................15
4.3 Topic-ontologies, Taxonomies, and Subject Hierarchies ............................17
**5 The Architecture of PROTON..................................................................20**
5.1 PROTON System Module Coverage ...........................................................21
5.2 PROTON Top Module Coverage ................................................................24
5.3 PROTON Upper Module Coverage.............................................................24
5.4 PROTON Knowledge Management Module...............................................24
5.5 Naming Conventions ...................................................................................24
5.6 Current Status...............................................................................................24
**6 PROTON Top Module Definitions and Upper Module Branches ................25**
6.1 Object branch ..............................................................................................25
6.1.1 Agent class ..............................................................................................26
6.1.2 Person class ..............................................................................................27
6.1.3 Group and Organization classes...............................................................28
6.1.4 Location class............................................................................................28
6.1.5 Statement class..........................................................................................31
6.1.6 InformationResource and Document classes ...........................................32
6.1.7 Product class .............................................................................................40
6.1.8 Service class..............................................................................................40
6.1.9 CommercialOrganization class ................................................................40
6.2 Happening branch .......................................................................................41

6.2.1 Event class ....... 42
6.2.2 Situation class ....... 42
6.2.3 TimeInterval class, modelling of time ....... 42
6.2.4 JobPosition class and modelling ....... 43
6.2.5 Role class ....... 45
6.2.6 Meeting class ....... 46
6.2.7 Project class ....... 46
6.2.8 OfficialPosition class ....... 46
6.2.9 Employee class ....... 46
6.2.10 Leader class ....... 46
6.2.11 Date class ....... 47
6.3 Abstract branch ....... 48
6.3.1 Number class ....... 48
6.3.2 ContactInformation class ....... 49
6.3.3 Language class ....... 49
6.3.4 Topic class and modelling ....... 49
6.3.5 BusinessAbstraction ....... 52
6.3.6 IndustrySector class and modelling ....... 53
6.3.7 TemporalAbstraction ....... 53
6.3.8 GeneralTerm ....... 54
6.3.9 NaturalPhenomenon ....... 54
6.3.10 SocialAbstraction ....... 54
**7 PROTON Knowledge Management Module Classes ....... 55**
7.1 protonkm:InformationSpace ....... 55
7.2 protonkm:SoftwareAgent ....... 55
7.3 protonkm:User ....... 56
7.4 protonkm:Profile ....... 56
7.5 protonkm:InformationSpaceProfile ....... 56
7.6 protonkm:UserProfile ....... 56
7.7 protonkm:Mention ....... 57
7.8 protonkm:WeightedTerm ....... 58
7.9 protonkm:Device ....... 58
**8 Relations to Other Standards ....... 59**
8.1 Dublin Core ....... 59
8.2 Wordnet ....... 59

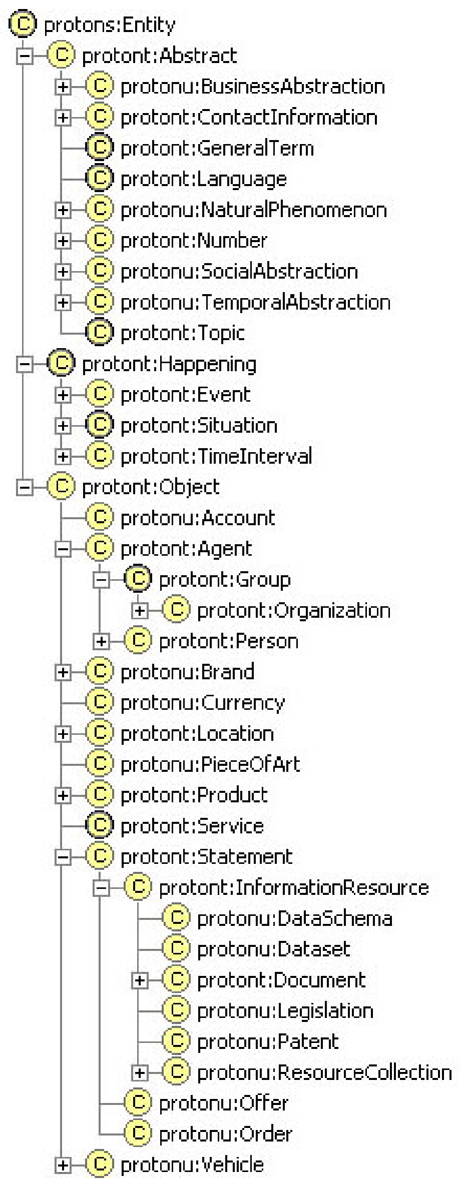
8.3 Alexandria Digital Library and GNS ........ 59
8.4 OpenCyc ........ 60
**9 Usage and Extension Guidance ........ 61**
**10 Future Development and Community Process ........ 64**
10.1 Community Process ........ 64
10.2 Refinement of protont:Location ........ 64
10.3 Remodelling of protonu:Employee ........ 65
10.4 Slimming down multiple-inheritance classes ........ 65
10.5 Remodelling of protons:EntitySource and protons:generatedBy ........ 65
10.6 Remodelling of other System module entities ........ 65
10.7 Remodelling of protonu:PublicCompany ........ 65
10.8 Conversion of protonu:hasUniversity ........ 66
**11 Conclusion ........ 67**
**12 References ........ 69**
**Appendix A: PROTON-specific Axioms ........ 70**

## 3 Introduction

An important practical approach to ontology generation is the use of background or pre-existing knowledge. In the scope of the SEKT project, the major body of such background knowledge is a basic upper-level ontology called PROTON (PROTo ONtology). This ontology is also used for metadata generation (WP2); it is an important part of the overall knowledge modelling and integration strategy of SEKT, as outlined in [15].

PROTON is a development of the KIMO ontology, which was created and used in the scope of the KIM platform for semantic annotation, indexing, and retrieval. KIM itself is further developed in the scope of WP2 “Metadata generation”. A comprehensive overview of KIM can be found in [12]. The home page of the KIM platform is http://www.ontotext.com/kim.

PROTON ontology contains about 300 classes and 100 properties, providing coverage of the general concepts necessary for a wide range of tasks, including semantic annotation, indexing, and retrieval of documents. The design principles can be summarized as follows (i) domain-independence; (ii) light-weight logical definitions; (iii) alignment with popular standards; (iv) good coverage of named entities and concrete domains (i.e. people, organizations, locations, numbers, dates, addresses). The ontology is originally encoded in a fragment of OWL Lite.



```
protons:Entity
  protont:Abstract
    protonu:BusinessAbstraction
    protont:ContactInformation
    protont:GeneralTerm
    protont:Language
    protonu:NaturalPhenomenon
    protont:Number
    protonu:SocialAbstraction
    protonu:TemporalAbstraction
    protont:Topic
  protont:Happening
    protont:Event
    protont:Situation
    protont:TimeInterval
  protont:Object
    protonu:Account
    protont:Agent
      protont:Group
        protont:Organization
      protont:Person
    protonu:Brand
    protonu:Currency
    protont:Location
    protonu:PieceOfArt
    protont:Product
    protont:Service
    protont:Statement
      protont:InformationResource
        protonu:DataSchema
        protonu:Dataset
        protont:Document
        protonu:Legislation
        protonu:Patent
        protonu:ResourceCollection
      protonu:Offer
      protonu:Order
    protonu:Vehicle
```

**Fig. 1. A representative view of the PROTON class hierarchy**

PROTON provides no specific support for general concepts, which are not likely to have named instances of theirs, such as ‘chair’ or ‘love’. However, the top level of the ontology is designed so as to allow easy extension in this direction.

In order to meet the requirements of the usage scenarios and to assure easy and gradual understanding, PROTON is separated into four modules:

- System module – it contains a few meta-level primitives (5 classes and 5 properties). It introduces the notion of 'entity', which can have aliases. The primitives at this level are usually the few things that have to be hard-coded in ontology-based applications. This module can be considered an application ontology. Within this document and in general, the System module of PROTON is referred to via the "`protons:`“ prefix.

- Top module – the highest, most general, conceptual level, consisting of about 20 classes. These ensure a good balance of utility, domain independence, and ease of understanding and usage. The top layer is usually the best level to establish alignment to other ontologies and schemata. Within this document and in general, the Top module of PROTON is referred to via the "`protont:`“ prefix.
- Upper module – over 200 general classes of entities, which often appear in multiple domains (e.g. various sorts of organizations, a comprehensive range of locations, etc.). Within this document and in general, the Upper module of PROTON is referred to via the "`protonu:`“ prefix.
- KM (Knowledge Management) module – 38 classes of slightly specalized entities that are specific for typical Knowledge Management tasks and applications. The KM module is actually the former SKULO ontology [15], further developed and integrated into PROTON. Within this document and in general, the PROTON KM module is referred to via the "`protonkm:`“ prefix.

The rest of this section provides a discussion on the scope of PROTON, as well as a short introduction to ontologies in an attempt to make sure that the purpose and the benefits of using ontologies may become evident. This discussion is extended in the subsequent section 4 which provides the design rationales of PROTON.

## 3.1 How PROTON Relates to KIM and KIMO

The major developments of PROTON with respect to KIMO can be outlined as follows:

- PROTON is represented in OWL Lite (versus KIMO in RDFS);
- PROTON was broken down into modules (System, Top, Upper, KM);
- It is KIM-independent (the KIMO-related concepts were relocated to specific, separate modules);
- Some tuning took place in response to the requirements of SEKT (WP5 “Knowledge Access” and the digital library case studies, WP11).

The KIMO ontology will not be developed further. The latest versions of KIM (after 1.12) make use of PROTON, along with a couple of complementary, KIM-specific modules, as follows:

- KIMSO (KIM System Ontology) – it contains a few meta- or system-level primitives used by KIM; it is available at http://www.ontotext.com/kim/2004/12/kimso.
- KIMLO (KIM Lexical Ontology) – it contains some lexical resource-related concepts, mostly used to represent lexica required by the KIM information extraction sub-system. It is available at http://www.ontotext.com/kim/2004/12/kimlo.

Both KIMSO and KIMLO import the System Module of PROTON.

## 3.2 Scope, Coverage, Compliance

The extent of specialization of the ontology is partly determined on the basis of case studies within the scope of the SEKT project and on a research of the entity types in a corpus of general news (including political, sports, and financial ones). The distribution of the entity types, which are most commonly referred to, varies greatly across domains. As researched in [11], despite the difference of type distributions, there are several general entity types that appear in all corpora – **`Person, Location, Organization, Money`** (**`Amount`**), **`Date`**, etc. The proper representation and positioning of those basic types was one of the objectives, backing the design of PROTON, and this was accomplished, for the most part, at the level of PROTON Top module layer.

The rationale behind PROTON is to provide a minimal, nevertheless sufficient ontology, suitable for semantic annotation, as well as a conceptual ground for more general KM applications. Its predecessor – KIMO – was designed from scratch for the purposes of KIM; a number of upper-level resources inspired its creation and development: OpenCyc (section 8.4), Wordnet 2.0 (section 8.2), DOLCE [16], EuroWordnet Top, and others. In order to keep the ontology simple and easy to understand, it is preserved small and naïve with respect to a great number of philosophical, mathematical, and logical problems.

One of the objectives of the development of PROTON has been to make it compliant with Dublin Core (section 8.1), the ACE annotation types[2], and the ADL Feature Type Thesaurus (section 8.3). This means that although those are not directly imported (for consistency reasons), a formal mapping of the appropriate classes and primitives is straightforward, on the basis of (i) compliant design and (ii) formal notes in the PROTON glosses, which indicate the appropriate mappings. For instance, in PROTON, a **`protont:hasContributor`** property is defined, with a domain **`protont:InformationResource`** (section 6.1.6) and a range **`protont:Agent`** (section 6.1.1), as an equivalent of the **`dc:contributor`** element in Dublin Core[3]. The development philosophy of PROTON is to make it compliant, in the future, with other popular standards and ontologies, such as FOAF[4].

## 3.3 Ontologies and Knowledge Representation

Formal knowledge representation (KR) is about building models of the world, of a particular domain or problem, which allow automatic reasoning and interpretation. Such formal models are called ontologies and they can be used to provide formal semantics (i.e. machine-interpretable meaning) to any sort of information: databases, catalogues, documents, web pages, etc. Having a better "understanding" of the information, machines can process it in a much more efficient manner.

[2] The ACE (Automatic Content Extraction) is one of the most influencing Information Extraction programs, see http://www.itl.nist.gov/iad/894.01/tests/ace/. A set of entity types is defined within "The ACE 2003 Evaluation Plan" (ftp://jaguar.ncsl.nist.gov/ace/doc/ace_evalplan-2003.v1.pdf). Those are: Person, Organization, GPE (a Geo-Political Entity), Location, Facility.

[3] According to Dublin Core (8.1): "An entity responsible for making contributions to the content of the resource. Examples of a Contributor include a person, an organization, or a service. Typically, the name of a Contributor should be used to indicate the entity."

[4] The *Friend of a Friend* (FOAF) project is about creating a Web of machine-readable homepages describing people, the links between them and the things they create and do. See http://www.foaf-project.org/

Imagine, for instance, a typical database, populated with the information that John is a son of Mary. It will be able to "answer" just a couple of questions: Who are the sons of Mary? and Whose son is John? An ontology-based system could handle a much bigger set of questions, because it will be able infer that: John is a child of Mary (the more general relation); Mary is a woman; Mary is the mother of John (the inverse relation); Mary is a relative of John (a generalization of the inverse relationship); etc. Although seeming rather simple in the eyes of a human, the above facts would remain "invisible" to a typical database and to any other information system, because their models of the world are limited to data-structures of strings and numbers.

Unfortunately, building ontologies and specifying the formal semantics of the data could be an extremely slow, expensive, and error-prone task. A number of linguistic and statistical methods are put to use in order to automate the process and enable the wide-spreading of ontology-based systems (those are covered in WP1 and WP2).

On the other hand, ontologies are crucial for many natural language processing (NLP), knowledge discovery, and text mining tasks and the like. They are, at the same time, the source of common sense, required to support non-trivial analyses, and the periscope, necessary to interpret, understand, and make use of the results. Further, ontologies also play a role in the natural language generation tasks – it is impossible to generate a reasonable, redundancy-free text without a formal model of the domain, the context, and the reader. This is how KR and NLP, two of the most prominent AI disciplines, can live in synergy, supporting each other.

## 3.4 PROTON: Glimpses of a Philosopher's Stance[5]

Insights of contemporary philosophy, as far as the notion of an ontology is concerned, makes a clear distinction between logicalized ontologies and non-logicalized. The seeds that gave birth to the notion of a logicalized ontology are planted far back into the work of such classical philosophical thinkers like Kant (in "Critique of Pure Reason") and Hegel (especially in his fundamental "Science of Logic"). What is characteristic for a logicalized ontology is what is *essential* for logic: the preservation of identity via the maintenance of necessity, i.e. a logicalized ontology is such precisely because of the explication of an equivalent preservation of identity when meanings (features, signs) are related from one category/type/class/grouping onto another one. From this aspect, PROTON may be classified as a *logicalized ontology*.

Further, another dividing line in this direction is the one between ontologies that logicalize the *existence* of things (existing, though not necessarily possessing "essence", i.e. real meaning or manifestation or materialization in the world), and ones that logicalize the *essence* of things (out of everything existing, the entities that have essence). Actually, what is worth mentioning about the notion of a logicalized ontology is that the latter implicates not only the very orderliness of things in reality, but also the *ways* through which the established orderliness – one or many of them - in terra existential can be accomplished. Therefore, an existence-logicalized ontology is one that is used to deal with the established orderliness and that does not suggest means for any autonomous causing, changing, or re-defining that orderliness (e.g. via the provision of a way to define it through the essence of ontologized entities).

[5] Partially inspired by a series of causeries between the authors of this document and Mr. V. Dafov, a professor at the Faculty of Philosophy at the Sofia University "St. Climent Ohridski"; http://www.uni-sofia.bg/faculties/philosophy/index.html.

Thus, PROTON can certainly be lined up as an existence-logicalized ontology, since it clearly models a fairly strictly defined (status quo/well-established) orderliness (or, organization) of what *exists* (i.e. via instances of types), moulded into classes and sub-classes (alternatively - types and sub-types). The concept of subordination of the order of entities (i.e. the hierarchy) is a prerequisite for the claiming of a possibility to deal with entities situated at the highest levels of the tree of subordination.

Ontologies of logicalized existence (or existence ontologies) like PROTON do not allow much dynamism as concerns any sort of self-driven extension of the very genes of the ontology organism: i.e., although PROTON is extremely flexible in terms of remodelling and extension, it however has some hard-coded parts (like the System module) and - what is more important - a stemming principle of birth and growth, which has already cut in the pattern (flexible, but nevertheless fixed) directing the development symphony. Oddly enough, but the latter “constraint” actually serves rather well the objective: an easily-extensible, upper-level, simplistic ontology, which can be used by machines.

As an existence ontology, PROTON is not, and cannot be, fully compatible with any other ontology of this type. Its common-sense basis is, of course, quite an arbitrary claim to deal with. However, the diversity of world knowledge and essential primitives and artefacts, added to the variety of cultural, social, educational, and any other background of humans actually blur the horizon of hope from a purely philosophical point of view if one wants an ontology that is at the same time basic enough to serve as a fundament for other layers, comprehensive in its coverage of "knowledge", “compliant” with the common-sense of “everybody”, etc. Therefore, our general claims are that:

- PROTON is not “perfect”, but as long as it can serve its purposes, it is good enough;
- PROTON is facing a long and interesting course of further development, especially one that is to be conditioned by the needs and ideas of the growing community that would (and will) use it.

A great deal of the potential misinterpretation when it comes to changes and extensions of PROTON can be attributed to those - so frequent - cases of misunderstanding of class names and their “meaning”, which is quite often mistaken for a lack of consensus between the pseudo-conflicting “common-senses” of the debating parties. The root of this kind of problems is dug deep under the ancient ground of the dichotomy of subjectiveness versus objectiveness of (the interpretation of) world reality by man.

As it can be argued, anything in our conditional, civilized world, as we all know it, is influenced (and in many cases – created) by us, humans. After all, the end users of PROTON are also humans and therefore it is all about everyone’s personal cognition and perception of reality. The cognition (and the recognition) of what is universally “correct” and “true” for everybody (or, to go further, of what “objective reality” is) is an issue that has been facing human minds for tens of centuries. What is more, presently machines could not be programmed to “think” for, or “judge” over, matters that today’s human civilization has not yet reached consensus about.

The very pieces of information and association that a certain element of an ontology may present – directly or not – to the perceiver, are like a subsequent handful of coins in the money-box of acquired knowledge – one is just happy to slip them inside the box. However, too few people would really check the coins for authenticity – i.e. the out-and-out suspicion, and consecutive judgment, regarding the verity and trustworthiness of the information received, are all the subject of cognition: it is (in the most part) an unconscious act, and it greatly depends on the individual personality, background, and erudition of the perceiver. And yes, this is what a machine cannot possibly be expected to cope with. According to Chomsky[6], who introduced the term 13, to cognize means to denote a relation a person has to his or her knowledge. Actually, cognizing is said to differ very little from knowing in the ordinary sense, but there are some important features of cognizing that set it off from the standard conception of what it is to know something. Perhaps the most salient feature of cognizing is that it is a relation primarily – though apparently not exclusively – associated with implicit or unconscious knowing or knowledge. What distinguishes cognizing per se from ordinary knowing is that in many cases, what is cognized is inaccessible to consciousness.

## 3.5 Ontology Languages

According to the analysis of ontology and knowledge representation languages and formats in [6] and also by other authors, it becomes evident that the consensus, which exists beyond RDF(S), is quite delicate [1]. The latter is a well-established knowledge representation and interchange language across the Semantic Web community. The rich diversity of RDF(S) repositories, APIs, and tools mould a mature environment for the development of systems, which are grounded in an RDF(S) representation of their ontological and knowledge resources. Because of the common acceptance of RDF(S) in the Semantic Web community, it would be easy to reuse the ontology and KB, as well as to enrich them with domain-specific extensions. The new OWL standard, [5], offers a clear, relatively consensual, and backward-compatible path beyond RDF(S), but it is still lacking in adequate tool support as regards the arena behind the scene of OWL Lite. What is more (or, rather, “what is less”), even the scalability of the tools, providing OWL Lite reasoning, is unclear.

Our experience shows, [12], that RDF(S) provides a sufficient level of expressiveness for the basic purposes of light-weight ontology definition and entity description. The most obvious nice-to-have primitives (equality, transitive and symmetric relations, etc.) are well covered in OWL Lite – the simplest first level of OWL. So, we suggest that OWL Lite is used, having RDF(S) as a back-up option.

The current version of the ontology is encoded in OWL Lite. It can also benefit from rule-based extensions. A relevant example of this is the **`protons:transitiveOver`** (OWL annotation) property, introduced in the PROTON System module. It suggests that one property is transitive with respect to another one, i.e.:

```
<P1,transitiveOver,P2>, <X,P1,Y>, <y,P2,z> => <x,P1,y>
```

To provide a specific example:

```
<hasSubject,transitiveOver,subTopicOf>,
<Book1,hasSubject,AfricanLions>,
<AfricanLions,subTopicOf,Africa> => <Book1,hasSubject,Africa>
```

[6] Chomsky, Noam (1980, p.69). Rules and Representations. New York, Columbia University Press.

Axioms of this sort are fairly specific and, as concerns repositories, we know of one (among others, perhaps) that supports it - Sesame[7]. It is mentioned hereby for reasons of consistency of this guidance, and also because: it has been extensively tested within the KIM Platform project; it is scalable enough (to the best of our knowledge); we believe it provides an easy shortcut towards the handling of what should be developed in the future when a more comprehensive (and standard) rule language is to be used. See Appendix A at the end of this document for a list of the PROTON-specific (i.e. non-standard, custom) axioms.

The modelling (knowledge formalization) approach behind PROTON is presented in better detail in section 4.2, where we provide sufficient introduction to RDF(S) and OWL, limited to the extent in which they are used in PROTON.

## 3.6 Physical Address and Namespaces

Each of the modules of PROTON is maintained as a separate OWL ontology. The namespaces are defined as follows:

- PROTON System: `http://proton.semanticweb.org/2005/04/protons#`
- PROTON Top: `http://proton.semanticweb.org/2005/04/protont#`
- PROTON Upper: `http://proton.semanticweb.org/2005/04/protonu#`
- PROTON Knowledge Management:
  `http://proton.semanticweb.org/2005/04/protonkm#`

For reasons of consistency, the namespaces of PROTON will stay as defined above as longer as possible, notwithstanding the year and month of production cited in the URL-s. In cases of major updates and subsequent changes of the namespaces, the new ones will be duly announced to the community and reflected on the PROTON home website (see below).

The home website of PROTON is accessible at the physycal address **`http://proton.semanticweb.org/`** where up-to-date versions of this document and the ontology modules of PROTON are available.

[7] Sesame is an RDF(S) repository, the development of which is led by Aduna b.v. http://sesame.aidministrator.nl

# 4 Design Rationales

PROTON (PROTo ONtology) is developed by Ontotext Lab in the scope of the SEKT project as a light-weight upper-level ontology, which serves as a modelling basis for a number of tasks in different domains. To mention just a few applications: PROTON is meant to serve as a seed for ontology generation (new ontologies extracted as extensions of PROTON); it is further used for automatic entity recognition and more generally Information Extraction (IE) from text, for the sake of semantic annotation (metadata generation). In a nutshell, PROTON is designed as a general-purpose domain-independent ontology. The above mission statement pre-determines a couple of drawbacks:

- PROTON is (relatively) un-restrictive. In other words, the conceptualization behind it is a general one; the definitions are rather partial; the possible interpretations are not constrained to the degree in which it would be possible and desirable if it was a domain- or task-specific one. The latter limits the "predictive" power of the ontology.
- PROTON is naïve in many aspects, as for instance the conceptualization of space and time. This is partly because proper models for these aspects would require usage of logical apparatus which is beyond the limits acceptable for some of the tasks of interest (e.g. queries and management of huge datasets/knowledge bases). And partly because it is very hard to craft strict and precise conceptualizations, which are adequate for a wide range of domains and applications.

Having accepted these drawbacks, we put a couple of additional requirements towards PROTON, namely, to allow for (i) low cost of adoption and maintenance and (ii) scalable reasoning. The high-level goal is to make feasible the usage of ontologies and the related reasoning infrastructure as a replacement for the DBMS.

## 4.1 Ontologies as RDBMS Schema

Here we discuss formal ontologies modelled through knowledge representation (KR) formalisms based on mathematical logic (ML); there is a note on the so-called topic-ontologies in a subsection below. If we compare the ontologies with the schemata of the relational DBMS, there is no doubt that the former represent (or allow for representations of) richer models of the world. There is also no doubt that the interpretation of data with respect to the fragments of ML, which are typically used in KR, is computationally much more expensive as compared to interpretation towards a model based on relational algebra. From this perspective, the usage of the lightest possible KR approach, i.e. the least expressive logical fragment, is critical for the applicability of ontologies in a bigger fraction of the contexts in which DBMS are used.

On our view, what is very important for the DBMS paradigm is that it allows for management of huge amounts of data in a predictable and verifiable fashion. For a CS graduate, it is relatively easy to outlook and understand a relational database schema. The efforts for understanding and management of such a schema grows in a sort of linear dependency on its size. The same broad expert can easily predict, understand, and verify the results of a query, even on top of a datasets with millions or even billons of records. This is the level of control and manageability required for systems managing important data, related to the performance of enterprises and public services. And this is the requirement which is not well covered by the heavy-weight, fully-fledged, logically expressive knowledge engineering approaches. Even taking a trained knowledge engineer and a relatively simple logical fragment (e.g. OWL DL), it is too hard for the engineer to maintain and manage the ontology and the data, with the growth of the size of the ontology and the scale of the data. We leave the above statements without a proof, hoping that most of the readers share our observations and intuition[8].

## 4.2 Formalization (Knowledge Representation) Approach

PROTON is originally formalized in OWL, more precisely, in a fragment of OWL Lite – the least expressive dialect of the W3C standard for encoding ontologies. The epistemological model of OWL is derived from the one of RDF. The world is modeled in terms of resources – everything (a web page, a train, number 3, a particular football match, the EURO currency) is modeled as resource having a unique identified (URI). The resources can “belong” or be instances of classes. The resources are described through triples of the form **`<subject,predicate,object>`**, which connect the resource (the subject), through a specific property (the predicate, the type of the relation) to the object, which could be either another resource, or a literal. The literals represent concrete data values (such as strings and numbers), essentially XML literals.

Thus, a person can be described with a couple of statements as follows:

- **`<#p1,type,#Person>`** - determines the class of the resource;
- **`<#p1,#hasFather,#p2>`** - connecting the person to the resource representing her father;
- **`<#p1,#hasBirthDate,“14/11/1927”>`** - connecting the person with the literal which represents her birth date.

PROTON defines a number of classes and properties. All classes which represent categories of objects or phenomena in the domain of discourse (generally, the world) are defined as sub-classes of **`Entity`**. We introduce the **`Entity`** class in order to distinguish the classes used to encode the proposed conceptualization from the others which have auxiliary and/or technical role in the RDF(S) vocabulary (e.g. **`rdfs:Class`** and **`rdf:Property`**, which are meta-modelling primitives). PROTON itself, also introduces few auxiliary classes and properties, defined, as a part of its System module, which are again detached from the “true” classes under **`Entity`**.

Below we will briefly present the RDFS and OWL constructs which are used to define the classes and properties in PROTON. We list all the modelling primitives, used in PROTON, but this is just a fraction of the full vocabulary of OWL and even of OWL Lite. Here follows the list of primitives:

- Both classes and properties has identifiers (URIs), titles (**`rdf:label`**) and descriptions (**`rdf:comment`**);

[8] We are tempted to share a hypothesis regarding the source of the unmanageability of any reasonably complex ML theory. It is our understanding that the ML provides a rough approximation for the process of human thinking, which renders it hard to follow. The relational algebra is also a rough approximation, but it seems simple enough, to be “simulated” by a trained person

- Classes can be defined as sub-classes, i.e. specializations, of other classes (via **`rdf:subClassOf`**). This means that all instances of the class are also property of its super class. For instance, **`City`** is a sub-class of **`Location`**. All classes are defined as instances of **`owl:Class`**, because in OWL the notion of class is more restrictive and clear as compared to RDF(S)).
- The properties are distinguished into object- and data-properties (respectively, **`owl:ObjectProperties`** and **`owl:DataProperties`**). The object-properties are referred to in some modelling paradigms as (binary) relations – they relate entities to other entities. The data-properties are often referred to as attributes – they relate entities to literals.
- Domain and ranges of properties are defined. The domain (**`rdfs:domain`**) specifies the classes of entities to which this property is applicable, i.e. which can be described through statements with this property as a predicate. The range (**`rdfs:range`**) specifies the classes of entities (for object-properties) or the datatypes (in case of data-properties). For instance, the property **`hasSister`** has the class **`Person`** as its domain and **`Woman`** as its range.
- Properties can be defined as sub-properties, i.e. specializations, of other properties (via **`rdf:subPropertyOf`**). Imagine that there is a couple of properties, **`p1`** and **`p2`**, for which **`<p1,subPropertyOf,p2>`**. The formal meaning is that for all pairs for which **`p1`** takes place, i.e. **`<x,p1,y>`**, **`p2`** also takes place, i.e. **`<x,p2,y>`** is also true. The hierarchy of family relationships provides a number of intuitive examples in this direction, e.g. **`hasSister`** is a sub-property of **`hasSibling`**, which on its turn is a sub-property of **`hasRelative`**.
- Properties can be defined as a symmetric (via **`owl:SymmtericProperty`**) and transitive (via **`owl:TransitiveProperty`**) ones. If **`p1`** is a symmetric property than whenever **`<x,p1,y>`** takes place, **`<y,p1,x>`** also takes place. If **`p2`** is a transitive property and **`<x,p2,y>`** and **`<y,p2,z>`** then it can be concluded that **`<x,p2,z>`** is also true. **`hasRelative`** is an obvious example for a property which is both symmetric and transitive.
- Object-properties can be defined to be inverse to each other (via **`owl:inverseOf`**). This means that if **`<p1,inverseOf,p2>`** than, whenever **`<x,p1,y>`** takes place, than **`<y,p2,x>`** can be concluded and vice versa.

As a design guidance, PROTON follows the principle of strict layering, which means that no classes or properties are instance of other classes[9]. We had limited ourselves to the above modelling means and principles for the following reasons:

- This level of expressivity allows for straightforward reasoning and interpretation on the basis of standard extensional semantic. Thus, it allows for usage of efficient reasoning algorithms and a variety of optimizations.
- It matches the modelling paradigm of the object-oriented approach (the OO-databases, languages such as Java, etc.). The major extensions to the OO model are the property inheritance, and the possibility to define transitive,

[9] The later (class as an instance of another class) is allowed by RDFS and OWL Full, but it changes the class of complexity of the logical fragment, which makes impossible the application of a range of efficient inference techniques.

symmetric and inverse properties. This ensures that the modelling style will be familiar to a wide audience within the CS community, including people without background in logic or KR.

- It allows easy maintenance, integration and migration to other formalizations.

## 4.3 Topic-ontologies, Taxonomies, and Subject Hierarchies

There is a wide range of applications for which the classification of different things (entities, files, web-pages, etc.) with respect to hierarchy of topics, subjects, categories, or designators has proven to be a good organizational practice which allows for efficient management, indexing, storage, or retrieval. Probably the most classical example in this direction are the library classification systems. Another typical representative are the taxonomies, which are widely used in the Knowledge Management field. Finally, Yahoo and DMoz, are popular and very large scale incarnations of this approach in the context of the world wide web.

As long as the above mentioned conceptual hierarchies represent a form of a shared conceptualization, it is not a surprise that they are also considered a sort of ontologies. It is our understanding, however, that these ontologies bear a different sort of semantics. The formal framework, which allows for efficient interpretation of DB-schema-like ontologies (such as PROTON) is not that suitable and compatible with the semantics of the topic hierarchies. For the sake of clarity, we introduce the term “schema ontology” to refer to the first and the term “topic ontologies” to refer to the second sort.

To allow for better understanding of the distinctions between topic- and schema-ontologies, we will provide here a brief sketch of the formal modelling of the semantics of the latter ones. The schema-ontologies are typically formalized with respect to the so-called extensional semantics, which in its simplest form allows for a two-layered set-theoretic model of the meaning of the schema elements. It can be briefly characterized as follows:

- The set of classes and relations on one hand is disjoint from the set of individuals (or instances), on the other. These form the vocabularies respectively of the TBox and the ABox in the description logics.
- The semantic of the classes is defined through the sets of their instances. Namely, the interpretation of a class is the set of its instances. The sub-class operation in this case is modeled as set inclusion (as in the classical algebraic set theory);
- The relations are defined through the sets of ordered n-tuples (the sequences of parameters or arguments) for which they hold. Sub-relations, are again defined through sub-sets. In the case of RDF/OWL properties, which are binary relations, their semantic is defined as a sets of ordered pairs of subjects and objects;
- This model can easily be extended to provide mathematical grounding for various logical and KR operators and primitives, such as cardinality constraints.

- Everything which cannot be modelled through set inclusion, membership, or cardinality within this model is undistinguishable or "invisible" for this sort of semantics – it is not part of way in which the symbols are interpreted.

This sort of semantics can be efficiently supported (in terms of induction and deduction algorithms) to a well-known extent. It can also be extended in different directions – something that the logicians are doing for centuries. A typical and very interesting representative of this class are the description logics, and in particular OWL DL.

It is our understanding that the semantic of the topics has a different nature. Topics can hardly be modelled with set-theoretic operations – their semanitc has more in common with the so-called intensional semantics. In essence, the distinction is that the semantic is not determined by the set of instances (the extension), but rather by the definition itself and more precisely the information content of the definition. The intensional semantic is in a way closer to associative thinking of the human being than the ML (in its simple encarnations) is. The criteria whether something is a sub-topic of something else have no much to do with the instances of the concrete class (if the topic is modeled this way). To some extent it is because the notion of instance is hard to define in this case.

Even disregarding the hypothesis for the different nature of the semantics of the topic-ontologies, we suggest that those should be kept detached from the schema-ontologies. The hierarchy of classes of the latter should not be mixed up with the topic hierarchies, because this can easily generate paradoxes and inconsistent ontologies. Imagine a schema-ontology, where we have definitions for **`Africa`** and **`AfricanLion`** – it is likely that **`Africa`** will be an instance of the **`Continent`** class and **`AfricanLion`** will be a sub-class of **`Lion`**. Imagine also a book classification – in this context **`AfricanLionSubject`** can be a subsumed by **`AfricaSubject`**. If we had tried to "re-use" for classification the definitions of **`Africa`** and **`AfricanLion`** from the schema-ontology, this would required that we define **`AfricanLion`** as a sub-class of **`Africa`**. The problems are obvious: one of this is not a class, and there is no easy way to redefine it so that the schema-ontology extensional sub-classing coincides with the relation required in the topic hierarchy. This example was proposed proposed by one of the authors to Natasha Noy for the sake of support of Approach 3 within the ontology modelling study [14]. One can find there some further analysis on the computational complexity implications of different approaches for modelling of topic hierarchies.

Based on the above arguments, we drew up a couple of principles and implemented those in PROTON:

- The class hierarchy of the schema ontology should not be mixed up with topic hierarchies;
- We should avoid precise and comprehensive modelling of the semantics of topics within a computational environment based on extensional semantics.

The **`Topic`** class within PROTON is meant to serve as a bridge between to two sorts of ontologies. The specific topics should be defined as instances of the Topic class (or a sub-class of it). The topic hierarchy is build using the **`subTopic`** property as a subsumption relation between the topics. The latter is defined to be transitive, but what is most important, it has nothing in common with the **`rdfs:subClassOf`** meta-property.

## 5 The Architecture of PROTON

PROTON is organized in three levels (including four modules), as shown on Fig. 2. The System ontology module occupies the first, basic layer; then the Top, and Upper, and KM ontology modules are upgraded on top of it to form the diacritical modular architecture of PROTON.

The System module is an application ontology level, which defines several notions and concepts of a technical nature that are substantial for the operation of any ontology-based software, such as semantic annotation and knowledge access tools. It includes the class **`protons:Entity`** – the top ("master") class for any sort of real-world objects and things, which could be of interest in some areas of discourse. In the system ontology it is defined that entities (i.e. the instances of **`protons:Entity`**) could have multiple names (instances of **`protons:Alias`**), that information about them could be extracted from particular **`protons:EntitySource`**-s, etc.

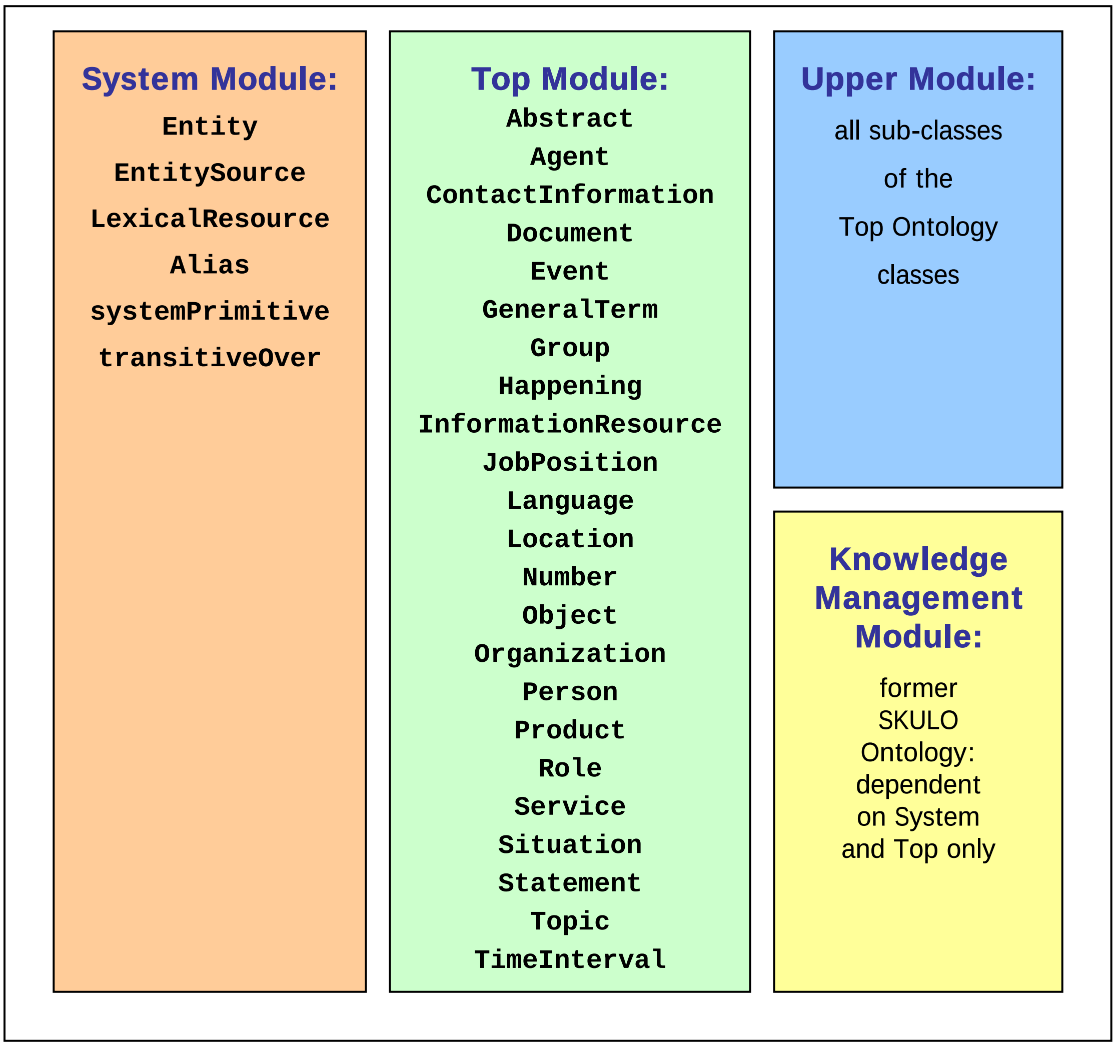


**Fig. 2. PROTON (PROTo ONtology) Modules**

The Top ontology module - the most essential ingredient - starts with some basic philosophically-reasoned distinctions between entity types, such as **`protont:Object`** – existing entities, as agents, locations, vehicles; **`protont:Happening`** – events and situations; **`protont:Abstract`** – abstractions that are neither objects nor happenings.

Those are further specialized by real-world, substantially real entity types of general importance: meetings, military conflicts, employment (job) positions, commercial, government, and other organizations, people, and various locations. It also covers numbers, time, money, and other specific values. Also, the featured entity types have their characteristic attributes and relations defined for them (e.g. **`protont:subRegionOf`** property for **`protont:Location-`**s, **`protont:hasPosition`** for **`protont:Person-`**s, **`protont:locatedIn`** for **`protont:Organization-`**s, **`protont:hasMember`** for **`protont:Group-`**s, etc.).

Further up-level, PROTON extends into its third layer, where either of two independent ontologies, which defines much more specific classes, can be used: PROTON Upper module or PROTON KM (Knowledge Management) module. Such examples are: **`protonu:Mountain`**, as a specific type of **`protont:Location`** (section 6.1.4); **`protonu:ResourceCollection`** as a sub-class of **`protont:InformationResource`** (section 6.1.6); **`protonkm:User`** (section 7.3) as a sub-class of **`protont:Agent`** (section 6.1.1). Having this ontology as a basis, one could easily add domain-specific extensions to it and on the whole it is straightforward to mould according to the specific requirements of the particular application, ontology, or KM tool that is to use it.

Lastly (although somewhat off-topic, however for the sake of laying down the last strokes on the canvas) perhaps it is a good idea to mention hereby the layers that are not part of PROTON anymore (but which used to be at the beginning). Actually, as already implied in section 3.1, a part of the KIMO ontology (the predecessor of PROTON) was specific to the KIM Platform, therefore all this information was taken out of the PROTON ontology and it was organized into two separate, KIM-specific modules: the KIM System Ontology[10] and the KIM Lexical Ontology[11]. The latter structures the information necessary for some symbolic information extraction systems (e.g. GATE), which in their turn are used in some semantic annotation systems (e.g. KIM). It is important for such systems, whereas it is not relevant to most of the applications that do not perform any extraction or annotation.

## 5.1 PROTON System Module Coverage

The System module of PROTON provides a sort of high-level system- or meta-primitives, which are likely to be accepted and even hard-coded in particular tools that may use PROTON. It is the only component in PROTON that is not to be changed for the purposes of ontology extension. The System module of PROTON includes the following classes (see Fig. 3. ): **`protons:LexicalResource; protons:Alias; protons:EntitySource; protons:Entity`**.

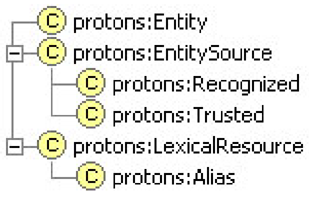


**Fig. 3. PROTON System Module class hierarchy**

The **`protons:Entity`** branch represents the root of the "true" ontology (the variety of entity classes) – it is the super-class of the PROTON Top module (section 6), while the other branches could be considered auxiliary ones. A property, related to the

[10] http://www.ontotext.com/kim/2004/12/kimso
[11] http://www.ontotext.com/kim/2004/12/kimlo

newly integrated layer of PROTON – the knowledge management module (section 7), is the **`protonkm:refersInstance`**, which provides a relation between **`protons:Entity`** and **`protonkm:Mention`** (section 7.7). **`protonkm:refersInstance`** is a property that is proprietary to **`protonkm:Mention`**.

The instances of the **`protons:EntitySource`** super-class are used to separate the trusted (pre-populated) information in the KB, from the one that is extracted automatically. Such a distinction is indicated by the **`protons:generatedBy`** property of the specific entity. This is done via the following two sub-classes:

- **`protons:Recognized`** - it serves to identify a source (like a program or a module) that is able to recognize and generate new entities from a text as part of IE or data mining tasks, which might be relevant to SEKT-related and other use cases (e.g. the British Telecommunications digital library case study within SEKT). Typically, those entities are not checked for correctness, consistency, relevance, etc., and therefore they are not trustable (the NE-recognition process in KIM/GATE can serve as a working example of this); and

- **`protons:Trusted`** – it may be used to indicate entities, imported from "trusted" sources, like GNS (section 8.3), World Fact Book, GATE/MUSE/KIM gazetteers, but also any other source that may be counted on to provide "trusted" information in terms of its correctedness in some universal sense of "being true".

In the long run, the key role of **`protons:EntitySource`** is to record the provenance of the instance data; moreover, the Semantic Web community is aware of systems, which are not related to SEKT and which also use similar primitives to record the provenance of instance data when knowledge is derived from multiple sources (one such example is the Flink[12] system). Actually, each instance of **`protons:Entity`** is linked to an instance of **`protons:EntitySource`** via the **`protons:generatedBy`** relation.

The **`protons:LexicalResource`** super-class is mostly dedicated to the encoding of various data (such as company suffixes like "AG" and "Ltd.", first names of persons, etc), related to IE and data mining processes. Such an approach is expected to prove useful for use cases within (and outside) the SEKT project. All the branches that used to be located "under" the **`protons:LexicalResource`** super-class, except for **`protons:Alias`**, were moved to, and are now considered part of, the KIM-specific KIM Lexical Ontology that is not a subject of this discussion. However, it is worth noting that two new sub-classes of **`protons:LexicalResource`** were recently added: **`protonkm:Mention`** (section 7.7) and **`protonkm:WeightedTerm`** (section 7.8), which are part of the newly integrated PROTON Knowledge Management module (section 7).

**`protons:Alias`** is an important class within this branch, representing the names of the instances of the **`protons:Entity`** class.

The **`protons:hasAlias`** relation is used to link an **`protons:Entity`** to its alternative names. Actually, it denotes an alias or a lexicalization of a concept - in most cases we are speaking not of a general term but rather of an alternative name of something.

[12] http://prauw.cs.vu.nl:8080/flink/

Specific names, such as “John” and “Smith”, are not aliased on themselves; however, "John Smith" and "Mr. Smith" could be. The most frequently used lexicalization of an entity is referred to by the **`protons:hasMainAlias`** property.

The high-level system- or meta-primitives and properties, contained in the PROTON System module, as shown in Fig. 4. below, are as follows:

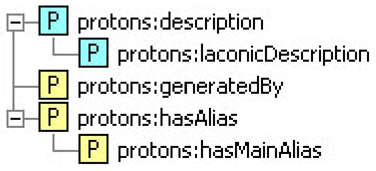


**Fig. 4. PROTON System module primitives and properties**[13]

- **`protons:description`** – this property denotes a textual description of an entity, usually a free text in some natural language; as defined in Dublin Core (see 8.1) for **`dc:InformationResources`**; in a sense, it is a specialization of **`rdf:comment`**.
- **`protons:laconicDescription`** – denotes an extremely short (typically, a single sentence) description of an entity; a sub-property of **`protons:description`**.
- **`protons:generatedBy`** – this property identifies the party that introduced the entity into the respective knowledge base; it relates the **`protons:Entity`** and **`protons:EntitySource`** super-classes of the PROTON system module.
- **`protons:hasAlias`** – this property refers an alias of the entity; it relates the system classes **`protons:Entity`** and **`protons:Alias`**.
- **`protons:hasMainAlias`** – refers the official (or otherwise most important) alias of the entity; a sub-property of **`protons:hasAlias`** property.
- **`protons:systemPrimitive`** - the system classes and properties are used to encode system specific information; those, as well as their instances, and related information, should usually not be presented to the end-user; in practice, user-interface and visualization modules can filter such primitives.
- **`protons:transitiveOver`** - it suggests that one property is transitive with respect to another one, thus making the modelling of a specific, but rather useful modelling pattern, possible; the semantics is defined with the following axiom (also available in **Appendix A** at the end of this document):

```
(p,transitiveOver,q) (x,p,y) (y,q,z) => (x,p,z).
```

An usage example of **`protons:transitiveOver`** follows:

```
(locatedIn, transitiveOver, subRegionOf)
(Ontotext,locatedIn,Bulgaria)
```

[13] Legend: The items coloured in yellow on all figures of this type throughout this document have a domain and a range, while the blue ones do not have a range specified.

```
(Bulgaria,subRegionOf,Europe) =>
(Ontotext,locatedIn,Europe)
```

## 5.2 PROTON Top Module Coverage

The PROTON Top module represents the most general classes, required by the SEKT case studies, as shown in Fig. 5. Those classes are discussed in more detail in section 6.

## 5.3 PROTON Upper Module Coverage

The PROTON Upper module is an extension of the Top module - roughly speaking, the sub-class branches of the Top module classes, as well as the corresponding properties and axioms. The major branches are discussed in section 6.

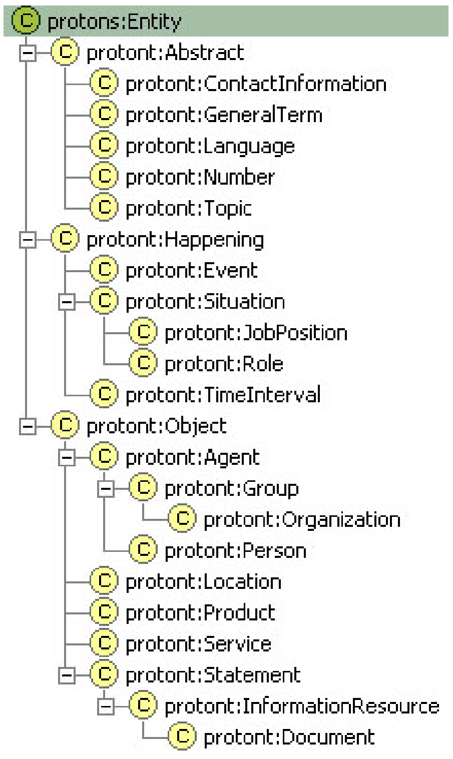


**Fig. 5. PROTON Top module classes**

## 5.4 PROTON Knowledge Management Module

The PROTON Knowledge Management Module contains the former SKULO ontology [15]. Its classes and properties are described in detail in section 7.

## 5.5 Naming Conventions

The naming conventions in PROTON, concerning the classes, the relations, and the attributes in the ontology, are as follows: the label of a class is composed of one or more words, written with capital first letters for each of the words, and without any intervals or alphanumeric symbols between them (in case a class label is a two-word one, for instance `protont:InformationResource`). The labels of relations and attributes follow the same rule, except for the non-capital first letter of the relation/attribute (e.g. `protont:locatedIn`).

## 5.6 Current Status

PROTON is in a process of constant development and improvement on the basis of the feedback regarding its usage within different projects and scenarios. The appropriate “community process” for collaborative discussion and development is to be realized within a mid-term time schedule.

# 6 PROTON Top Module Definitions and Upper Module Branches

The top module of PROTON contains some general classes, which will be used in the three SEKT case studies, the knowledge discovery, metadata generation, and intelligent knowledge access tools. These top-level classes represent the most common, basic, and inclusive notions of world knowledge. This section describes in detail the PROTON Top module super-classes, plus some of their more significant sub-classes, belonging to the PROTON Upper module.

The design at the highest level of the Top module follows the stratification principles of DOLCE, [16] through the establishment of the PROTON trichotomy of Objects (**`dolce:Endurant`**), Happenings (**`dolce:Perdurant`**), and Abstracts (**`dolce:Abstract`**).

## 6.1 Object branch

**`protont:Object`**-s (a PROTON Top module class) are entities that could be claimed to exist (in some sense of existence). An object can play a certain role in some happenings. Objects could be substantially real (as the Buckingham Palace or a hardcopy book) or substantially imperceptible (say, an electronic document – it only "exists" virtually, one cannot touch it). The proprietary relations and attributes of **`protont:Object`** are:

- **`protont:hasContactInfo`** – this property allows for relations between the **`protont:Object`** and **`protont:ContactInformation`** (6.3.2) (a branch of **`protont:Abstract`**).
- **`protont:isOwnedBy`** – this property relates a particular organization to the agents, which are members of that organization. This predicate indicates a `generic' type of membership, although there may be specialized kinds of membership in the same organization.

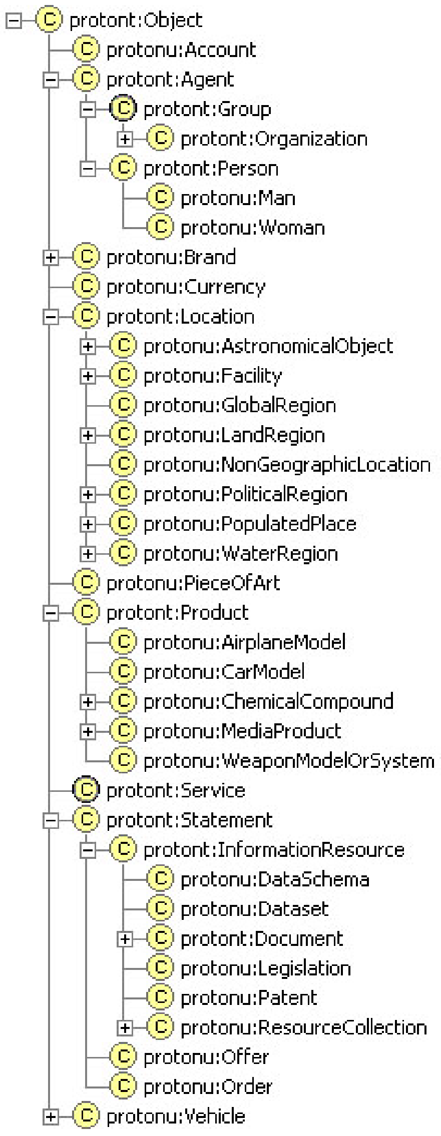


**Fig. 6. protont:Object class hierarchy**

Typically, membership eligibility is determined by the organization and accepted with the agent's voluntary affiliation. In many cases **`protont:Person`**-s that take **`protont:JobPosition`**-s within an **`protont:Organization`** are considered members of the organization, although this is not encoded here formally in any way.

The **`protont:Object`** top module class is a super-class of the following Top module classes (sub-sections in brakets): **`protont:Agent`** (6.1.1), **`protont:Person`** (6.1.2),

**protont:Group** (6.1.3), **protont:Organization** (6.1.3), **protont:Location** (6.1.4), **protont:Statement** (6.1.5), **protont:InformationResource** (6.1.6), **protont:Document** (6.1.6), **protont:Product** (6.1.7), and **protont:Service** (6.1.8).

### *6.1.1 Agent class*

**protont:Agent** (a PROTON Top module class) is something, which can show (carry out) an independent action, whether consciously or not. Most animals are considered agents, in most contexts; so are most organizations. According to DOLCE, [16], agents are "objects to which we ascribe intentions, beliefs, and desires". **protont:Agent** here also denotes any automatic services, including web services and servers.

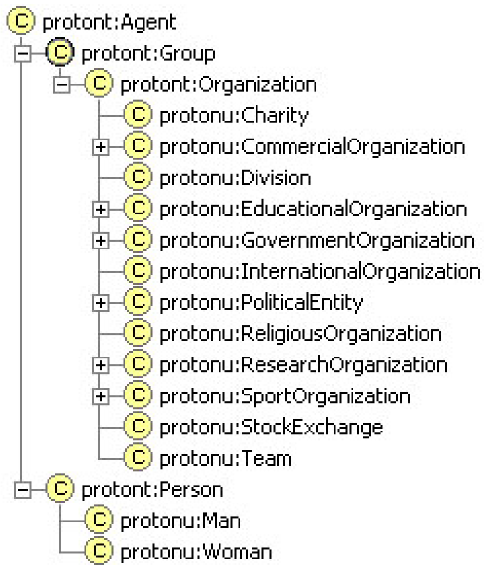

protont:Agent
protont:Group
protont:Organization
protonu:Charity
protonu:CommercialOrganization
protonu:Division
protonu:EducationalOrganization
protonu:GovernmentOrganization
protonu:InternationalOrganization
protonu:PoliticalEntity
protonu:ReligiousOrganization
protonu:ResearchOrganization
protonu:SportOrganization
protonu:StockExchange
protonu:Team
protont:Person
protonu:Man
protonu:Woman

**Fig. 7. protont:Agent class hierarchy**

Apart from the properties, inherited from the **protont:Object** super-class, **protont:Agent** has three proprietary ones, as follows:

- **protont:involvedIn** – this property allows for various relations between **protont:Agent** and members of the **protont:Happening** branch, for instance.

- **protont:isLegalEntity** – this property determines whether a particular **protont:Agent** is a legal entity or a non-legal one. Its range should be constrained to ***Boolean***. **protont:Agent-**s, for which the value is ***True***, correspond to instances of http://www.cyc.com/2003/04/01/cyc#LegalAgent, which is defined as follows "Each instance of #$LegalAgent is an agent who has some status in a particular legal system. At the very least, such an agent is recognized by some legal authority as having some kinds of rights and/or responsibilities as an agent (e.g., #$citizens of Germany), ...". In PROTON, it is modelled as a property in order to avoid multiple inheritance of classes and/or multiple classifications of instances.

  This property is quite specific and its inclusion as a property of **protont:Agent** was triggered by practical considerations, as the legal/non-legal connotation for an agent. Because of the relation it provides, it allows the user of a respective application, using the ontology, to narrow his/her search to a great extent. For instance, if one searches for documents, containing the company name *Morgan Stanley*, and they search for a **protont:Agent**, the name of which begins with *Morgan*, then the potential search result set would include both company names and proper names (e.g. the “investment banking giant” *Morgan Stanley* and Hollywood actor *Morgan Freeman*); while if the search is specialized further via the **protont:isLegalEntity** property, the search should only return documents containing the respective "legal" entities, i.e. for the purposes of this example - mostly company names.

- **protont:partiallyControls** - any sort of partial control of an agent with respect to an object; this intransitive relation provides for dependency relations between **protont:Agent**-s and **protont:Object**-s.

The sub-classes of **protont:Agent** are both PROTON Top module and PROTON Knowledge Management module (section 7) classes: **protont:Person**, **protont:Group**, and **protont:Organization** - described further in this document in separate sections (6.1.2, 6.1.3) – and **protonkm:InformationSpace**, **protonkm:SoftwareAgent**, and **protonkm:User -** described in sections 7.1, 7.2, and 7.3.

### *6.1.2 Person class*

**protont:Person** (a PROTON Top module class) is an agent (within the meaning of **protont:Agent** in PROTON), which is an individual who is a human being (i.e. any living or extinct member of the family Hominidae).

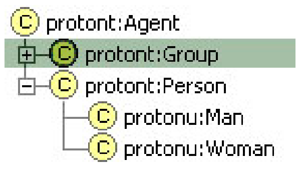


**Fig. 8. protont:Person class hierarchy**

Apart from the properties, inherited from the **protont:Object** and **protont:Agent** super-classes, **protont:Person** has four proprietary ones, as follows:

- **protont:hasPosition** – this property relates a **protont:Person** to a **protont:JobPosition** (a sub-class of **protont:Situation**: the situation of a person holding a position within an organization) – e.g. in the company he/she works for.

- **protont:hasProfession** – this property relates a **protont:Person** to a **protonu:Profession** (a sub-class of the **protonu:SocialAbstraction** class in the **protont:Abstract** branch).

- **protont:hasRelative** - this property relates a **protont:Person** to another one, who he is a relative to and who is a relation to the former. This is a many-to-many, bi-directional relationship. This relation has a number of specializations, presented in

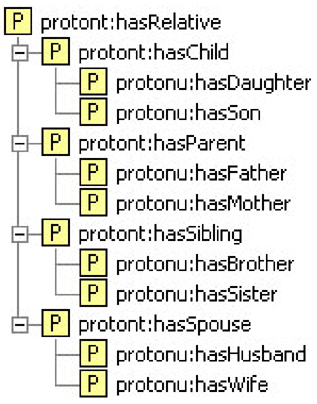


**Fig. 9. Hierarchy of Family Relations**

- **protont:isBossOf** – this property relates a **protont:Person** to another one, who he is an immediate boss or supervisor of. This is a many-to-many relationship, i.e. there can be more than one boss of a person, even co-temporally.

### *6.1.3 Group and Organization classes*

**`protont:Group`** (a PROTON Top module class) is a group of agents, which is not necessarily organized in any way. This could be the group of people within a bus or the shareholders of a company. **`protont:Organization`** (also a PROTON Top module class) is set as a sub-class of **`protont:Group`** because of this very difference in the presence or the absence of organization of the agents in the group. **`protont:Organization`** denotes a group, which is established in such a way that certain known relationships and obligations exist between the members, and/or between the organization and its members, and/or between the organization and `outsiders' (individuals or groups). The informal definition is adapted from OpenCyc (section 8.4). **`protont:Organization`** includes both informal and legally constituted organizations. Organizations can act as agents - to undertake projects, to enter into agreements, to own property, etc. Most organizations have names. Almost all have at least two members.

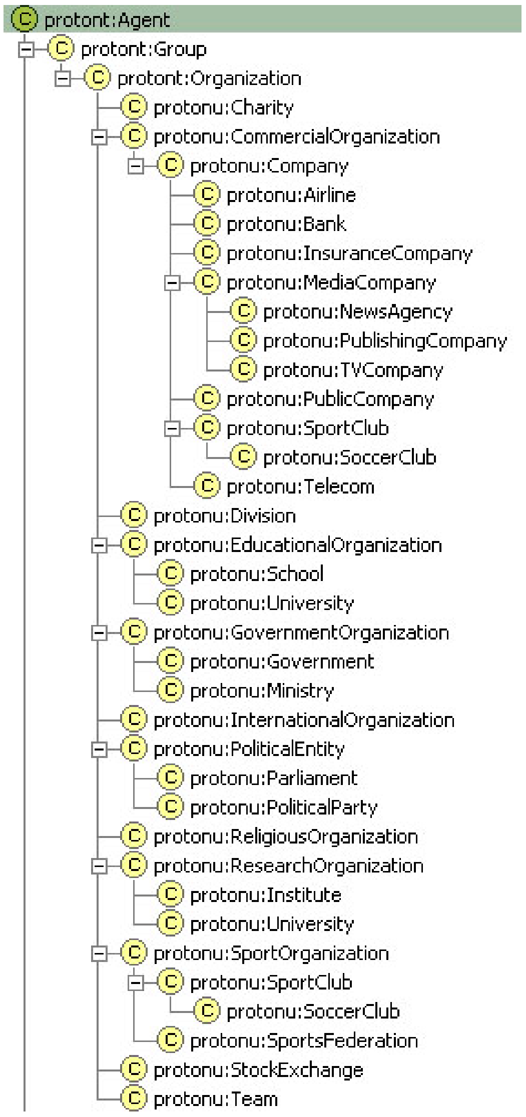
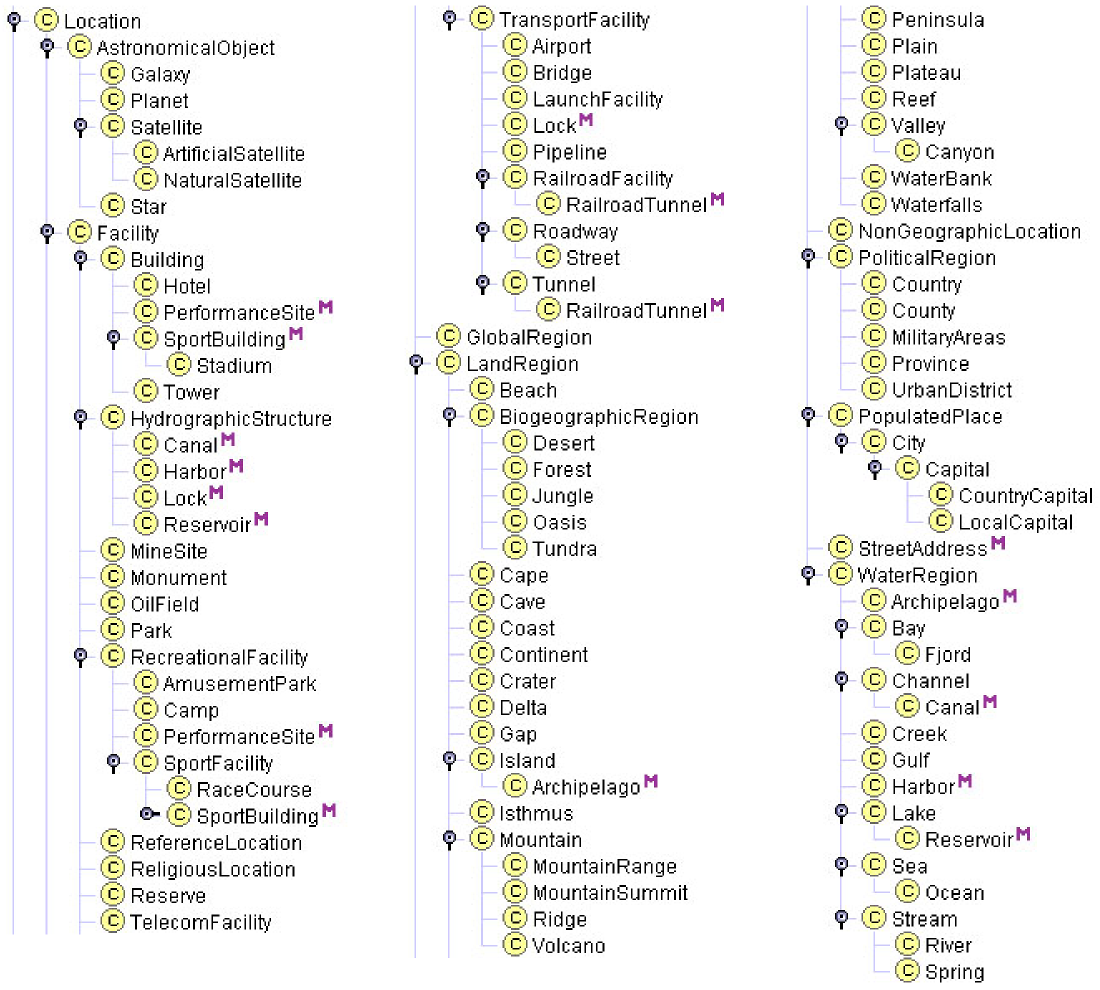


**Fig. 10. protont:Group and protont:Organization class hierarchy**

### *6.1.4 Location class*

**`protont:Location`** (a PROTON Top module class) is a sub-class of **`protont:Object`.** A location in PROTON is usually deemed a geographic location on the earth, however any sort of 3D regions also fit in this class. The **`protont:Location`** branch of PROTON contains over 100 sub-classes, aligned with the **Alexandria Digital Library Feature Type Thesaurus** and GNS[14] (see section 8.3). The sub-classes of the latter two, which are not included as specializations of **`Location`** in PROTON, are **Administrative areas** (those are to be classified directly as instances of **`Location`**), **Territorial waters**, and **Tribal areas**. For each specific type, the corresponding NIMA[14] GNS designators (DSG) are given.



```
Location
    AstronomicalObject
        Galaxy
        Planet
        Satellite
            ArtificialSatellite
            NaturalSatellite
        Star
    Facility
        Building
            Hotel
            PerformanceSite M
            SportBuilding M
                Stadium
            Tower
        HydrographicStructure
            Canal M
            Harbor M
            Lock M
            Reservoir M
        MineSite
        Monument
        OilField
        Park
        RecreationalFacility
            AmusementPark
            Camp
            PerformanceSite M
            SportFacility
                RaceCourse
                SportBuilding M
        ReferenceLocation
        ReligiousLocation
        Reserve
        TelecomFacility
        TransportFacility
            Airport
            Bridge
            LaunchFacility
            Lock M
            Pipeline
            RailroadFacility
                RailroadTunnel M
            Roadway
                Street
            Tunnel
                RailroadTunnel M
    GlobalRegion
    LandRegion
        Beach
        BiogeographicRegion
            Desert
            Forest
            Jungle
            Oasis
            Tundra
        Cape
        Cave
        Coast
        Continent
        Crater
        Delta
        Gap
        Island
            Archipelago M
        Isthmus
        Mountain
            MountainRange
            MountainSummit
            Ridge
            Volcano
        Peninsula
        Plain
        Plateau
        Reef
        Valley
            Canyon
        WaterBank
        Waterfalls
    NonGeographicLocation
    PoliticalRegion
        Country
        County
        MilitaryAreas
        Province
        UrbanDistrict
    PopulatedPlace
        City
            Capital
                CountryCapital
                LocalCapital
    StreetAddress M
    WaterRegion
        Archipelago M
        Bay
            Fjord
        Channel
            Canal M
        Creek
        Gulf
        Harbor M
        Lake
            Reservoir M
        Sea
            Ocean
        Stream
            River
            Spring
```

**Fig. 11. The Location class (of the Top) and its branch (of the Upper module)**

The reason for the specialization of the **`protont:Location`** Top module class as a distinctive, broad-leaved branch of PROTON is that geographic features (**`Location`**-s) represent quite a thick piece of the cake of substantial entities of general importance. In a way, it is a rather autonomous kind of a sub-ontology within the framework of PROTON. Our goal was to include the most important and frequently used types of **`Location`**-s (which are specializations of **`protons:Entity`** through **`protont:Object`**), including relations between them. Such relations are **`protonu:hasCapital`**, **`protont:subRegionOf`** (a much more specific relation than the transitive **`protont:partOf`**, having the **`protons:Entity`** system module super-class as its domain), relations between **`protont:Location-`**s and other

[14] GNS – GEOnet Names Server, the GNS database of NIMA (National Imagery and Mapping Agency of United States). http://earth-info.nga.mil/gns/html/. See section 8.3.

**protons:Entity**-s (**Organization locatedIn Location**), and various other properties. The proprietary relations and properties for **protont:Location** are as follows:

- **protont:NIMA_GNS_DSG** – the designator of the entity according to the NIMA GeoNames Server.
- **protont:NIMA_GNS_UFI** – the Unique Feature Identifier from the NIMA GNS. A number that uniquely identifies the location.
- **protonu:hasUniversity** – a relation between **protont:Location** and **protonu:University** (a sub-class of **protont:Group**).
- **protont:latitude** – in degrees, minutes, and seconds: no sign (+) = North; negative sign (-) = South.
- **protont:longitude** – in degrees, minutes, and seconds: no sign (+) = East; negative sign (-) = West.
- **protont:subRegionOf** – the general part-of relation, which here designates a place between a whole and each of its parts. It has a number of specializations.

The **protonu:officialPositionIn** property has **protont:Location** as its range, its domain being **protonu:OfficialPosition**. Thus, by inheritance (as **protonu:OfficialPosition** is a sub-class of **protont:JobPosition** – see section 6.2.4), this property models the **protont:Person - protont:Location** relation.

Also, among the properties that **protont:Location** inherits from the **protont:Object** super-class, the most important one is **protont:locatedIn**. It is a transitive relation of great significance because it is the key property that ensures the relation between other **protons:Entity**-s and the **protont:Location** branch. Further, the **protont:subRegionOf** relation (described above) extends the “path” into a specialized ‘part-of’ type of a relation between different **protont:Location**-s.

The **protont:Location** entity denotes an area in the 3D space, which includes geographic entities with physical boundaries, such as geographical areas and landmasses, bodies of water, geological formations, and also politically defined areas (e.g. ”U.S. Administered areas”).

The classification hierarchy in the PROTON Upper module layer for **protont:Location** (consisting of 108 sub-classes of **protont:Location**) is based on the ADL Feature Type Thesaurus version 070203. The differences between the sub-classes zoom in on considerations of simplicity; a number of distinctions and unnecessary levels of abstraction were removed wherever they proved irrelevant to a general (non-geographic) context, as the ontology had to be moulded and kept to be easy to understand by an “average-level” user. The **protont:Location** class and its upper module sub-classes provide the following additional information:

- the exact type of a feature, e.g. to be able to recognize a geographic feature as an instance of **protonu:CountryCapital** instead of the more broad-sensed **protont:Location**;
- relations between a strictly geographic (spatial) feature and other entities (or at least different types of locations), e.g. ”Devils Tower” is a **protonu:Mountain**

(a strictly geographic feature) that is **`protont:locatedIn`** USA (a country, i.e. having much larger connotations than just a location on the planet), and it is **`protont:subRegionOf`** the state of Wyoming (i.e. administrative, political, etc. associations);

- the different names of a location (e.g. ”Peking” and ”Beijing” are two aliases for one and the same location);
- the transitive **`protont:subRegionOf`** relation allows one to search for Entities located in a continent (e.g. ”Morgan Stanley” - **`locatedIn`** - ”New York” – **`subRegionOf`** - ”NY” - **`subRegionOf`** - ”USA” – **`subRegionOf`** - ”North America”)
- **`protons:trusted`** vs. **`protons:recognized`** sources, linked by the **`protons:generatedBy`** property of a **`protont:Location`** is an extra hint in disambiguation tasks. The class hierarchy is shown on Fig. 11.

An interesting new relation where **`protont:Location`** is range, is provided by the property **`protonkm:hasLocation`** that relates the class with the PROTON Knowledge Management module (section 7) class **`protonkm:UserProfile`** (section 7.6).

### *6.1.5 Statement class*

**`protont:Statement`** (a PROTON Top module class) is a message that is stated or declared; a communication (oral or written) setting forth particulars or facts etc; "*according to his statement he was in London on that day*". Adapted from Wordnet (section 8.2).

The **`protont:Statement`** class has three proprietary properties, as follows:

- **`protont:statedBy`**: this property best matches the **dc:creator** element (an entity primarily responsible for making the content of the resource; examples of a Creator include a person, an organization, or a service). Via the **`protont:statedBy`** property, **`protont:InformationResource`** can be linked to the **`protont:Agent`** class.
- **`protont:validFrom, protont:validUntil`**: these properties define the range (period) of validity of the information resource. They align to a great extent to the **dc:value** element refinement, as defined in Dublin Core (section 8.1)

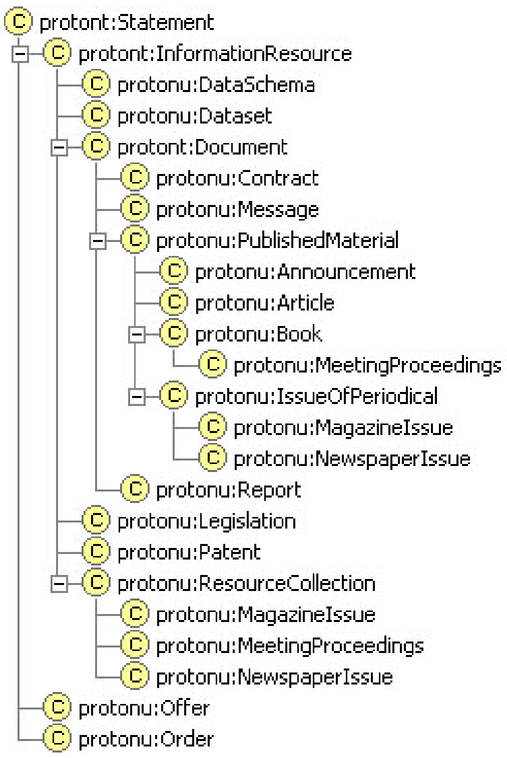


**Fig. 12. protont:Statement, protont:InformationResource, and protont:Document class hierarchy**

### *6.1.6 InformationResource and Document classes*

**`protont:InformationResource`** (a PROTON Top module class) denotes an information resource with an identity, as defined in Dublin Core (DC2003ISO, see section 8.1 for details). Its super-class is **`protont:Statement`**. Since **`protont:Statement`** is described as "a message that is stated or declared; a communication (oral or written) setting forth particulars or facts etc.", **`protont:InformationResource`** is considered any communication or message that is delivered or produced, taking into account the specific intention of its originator, and also the supposition (and anticipation) for a particular audience or counter-agent in the process of communication (i.e. passive or active feed-back). For instance, an **`protonu:Offer`** differs from an **`protonu:Announcement`** by the (non)obligatory aspect of an audience.

The **`protont:InformationResource`** branch contains another top class of the PROTON Top module - **`protont:Document`**. This is the information content of any sort of a document; any tangible (material) aspects of a document are ignored; it is usually a document in free text with no formal structure or semantics. **`protont:Document`** is a distinctive type of an information resource: what distinguishes the **`protont:Document`** sub-class, as a specific type of an information resource, are the aspects of intention and "particularism" of a document, at least in the "general" connotations and understanding about "what a document is" and what cannot be described as a document (e.g. a database is not a document, it is rather a dataset). Descriptions of the proprietary properties and attributes of **`protont:Document`** follow further in this section.

A new sub-class of **`protont:InformationResource`**, related to the newly integrated layer of PROTON – the knowledge managemend module (section 7), is the

**`protonkm:Profile`** class (section 7.4). Another **`protonkm`**-specific news is the **`protonkm:occursIn`** property that relates the **`protonkm:Mention`** (section 7.7) class to **`protont:InformationResource`**.

The **`protont:InformationResource`** branch was designed with tight adherence to the Dublin Core (DC, section 8.1) metadata elements set. In this connection, it is worth mentioning that **`protont:InformationResource`** is quite close in correspondence to what is considered **`dc:metadata`** in DC - of course, on the understanding that a high level of abstraction from the SEKT-specific meaning and usage of the term “metadata” is kept – the similarity is not in the senses of the two notions, but rather in the way they are described and defined “from the outside".

An alignment of **`protont:InformationResource`** and **`protont:Document`** top-ontology classes and branches in PROTON against Dublin Core (DC, section 8.1), follows:

**PROTON `InformationResource` classes and properties*, and DC-related comments**

*excluding the ones related to the knowledge management module (section 7)

**`protont:InformationResource`** (a PROTON Top module super-class):

An information resource with an identity, as defined in Dublin Core (DC2003ISO). An important distinction in PROTON is the differentiation between **`protont:InformationResource`** and **`protont:Document`**: information resources are all types of information resources that do not fall under the classification criteria of the **`protont:Document`** super-class.

Proprietary properties/attributes/relations of **`protont:InformationResource`** in PROTON (also valid for **`protont:Document`**) include:

- **`protonu:compliantWithSchema`**. PROTON description: links a dataset with a schema it complies to. DC-relevance: in part, this property corresponds to the **`dc:relation`** metadata element, as defined in DC, as regards the **`protonu:Dataset`** and **`protonu:DataSchema`** sub-classes.

- **`protont:derivedFromSource`**. PROTON description: a reference to a resource from which the present resource is derived. The present resource may be derived from the Source resource in whole or in part. Recommended best practice is to identify the referenced resource by means of a string or number conforming to a formal identification system. DC2003ISO. DC-relevance: **full equivalence** between the **`protont:derivedFromSource`** property in PROTON and the **`dc:source`** metadata element (labelled *Source*) in DC.

- **`protont:hasContributor`**. PROTON description: an entity responsible for making contributions to the content of the resource. Examples of a Contributor include a person, an organization, or a service. DC2003ISO. DC-relevance: **full equivalence** between the **`protont:hasContributor`** property in PROTON and the **`dc:contributor`** metadata element (labelled *Contributor*) in DC.

- **`protont:hasDate`**. PROTON description: typically, Date will be associated with the creation or the availability of the resource. Recommended best practice for encoding the date value is defined in a profile of ISO 8601 [W3CDTF] and includes (among others) dates of the form YYYY-MM-DD. For official documents, it could be the date of signature. DC-relevance: **full equivalence** between the **`protont:hasDate`** property in PROTON and the **`dc:date`** metadata element (labelled *Date*) in DC.

- **`protont:hasSubject`**. PROTON description: A topic of the content of the resource. Comment: typically, Subject will be expressed as keywords, key phrases, or classification codes that describe a topic of the resource. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme. (DC2003ISO, Subject). DC-relevance: **full equivalence** between the **`protont:hasSubject`** property in PROTON and the **`dc:subject`** metadata element (labelled *Subject and Keywords*) in DC.

- **`protont:inLanguage`**. PROTON description: A language of the intellectual content of the resource. Recommended best practice is to use RFC 3066 [RFC3066], which, in conjunction with ISO 639 [ISO639], defines two- and three-letter primary language tags with optional sub-tags. Examples include "en" or "eng" for English, "akk" for Akkadian, and "en-GB" for English used in the United Kingdom. DC2003ISO. This property relates the **`protont:InformationResource`** and **`protont:Language`** top module classes. DC-relevance: **full equivalence** between the **`protont:inLanguage`** property in PROTON and the **`dc:language`** metadata element (labelled *Language*) in DC.

- **`protont:informationResourceCoverage`**. PROTON description: The extent or scope of the content of the resource. Typically, Coverage will include spatial location (a place name or geographic coordinates), temporal period (a period label, date, or date range), or jurisdiction (such as a named administrative entity). Recommended best practice is to select a value from a controlled vocabulary (for example, the Thesaurus of Geographic Names [TGN]) and to use, where appropriate, named places or time periods in preference to numeric identifiers such as sets of coordinates or date ranges. DC2003ISO. DC-relevance: **full equivalence** between the **`protont:informationResourceCoverage`** property in PROTON and the **`dc:coverage`** metadata element (labelled *Resource Coverage*) in DC.

- **`protont:informationResourceIdentifier`**. PROTON description: An unambiguous reference to the information resource within a given context. Recommended best practice is to identify the resource by means of a string or number conforming to a formal identification system. Formal identification systems include but are not limited to the Uniform Resource Identifier (URI) (including the Uniform Resource Locator (URL)), the Digital Object Identifier (DOI), and the International Standard Book Number (ISBN). DC2003ISO. DC-relevance: **full equivalence** between the **`protont:informationResourceIdentifier`** property in PROTON and the

**dc:identifier** metadata element (labelled *Resource Identifier*) in DC.

- **protont:informationResourceRights**. PROTON description: Information about rights held in and over the resource. Typically, Rights will contain a rights management statement for the resource, or reference a service providing such information. Rights information often encompasses Intellectual Property Rights (IPR), Copyright, and various Property Rights. If the Rights element is absent, no assumptions may be made about any rights held in or over the resource. DC2003ISO. DC-relevance: **full equivalence** between the **protont:informationResourceRights** property in PROTON and the **dc:rights** metadata element (labelled *Rights Management*) in DC.

- **protont:resourceFormat**. PROTON description: The physical or digital manifestation of the resource. Typically, Format will include the media-type or dimensions of the resource. Format may be used to identify the software, hardware, or other equipment needed to display or operate the resource. Examples of dimensions include size and duration. Recommended best practice is to select a value from a controlled vocabulary (for example, the list of Internet Media Types [MIME] defining computer media formats). DC2003ISO. DC-relevance: **full equivalence** between the **protont:resourceFormat** property in PROTON and the **dc:format** metadata element (labelled *Format*) in DC.

- **protont:resourceType**. PROTON description: The nature or genre of the content of the resource. Type includes terms describing general categories, functions, genres, or aggregation levels for content. Recommended best practice is to select a value from a controlled vocabulary (for example, the DCMI Type Vocabulary [DCT]). To describe the physical or digital manifestation of the resource, use the resourceFormat property. DC2003ISO. DC-relevance: **full equivalence** between the **protont:resourceType** property in PROTON and the **dc:type** metadata element (labelled *Resource Type*) in DC.

- **protont:title**. PROTON description: A name given to an information resource. Typically, title will be a name by which the resource is formally known. (DC2003ISO, Title there). DC-relevance: **full equivalence** between the **protont:title** property in PROTON and the **dc:title** metadata element (labelled *Title*) in DC.

Further comments: the **protont:InformationResource** property set covers a major part of the Dublin Core Metadata Element Set. Some of the DC metadata elements **dc:relation**, **dc:publisher**, **dc:description**, and **dc:creator** correspond to the properties that the **protont:InformationResource** super-class inherits from the **protons:Entity** master-class, respectively:

| **PROTON** property, inherited by **protonu:InformationResource**: | Corresponding **Dublin Core** metadata element or element refinement: |
|---|---|
| **protons:description**: a textual description of an entity. Usually a free | **dc:description**: a textual description of an entity. Usually a free text in some |

| | |
|---|---|
| text in some natural language. As defined in DC2003ISO for **`protont:InformationResource`**-s. | natural language. As defined in DC2003ISO for Information Resources. |
| The property, which corresponds to the **`dc:creator`** element best, is the **`protont:statedBy`** property, inherited from the **`protont:Statement`** super-class (see the description of **`protont:Statement`** in the previous section for details). Via this property, **`protont:InformationResource`** can be linked to the **`protont:Agent`** class. The **`protons:generatedBy`** property (inherited from **`protons:Entity`**) could not play the role of a creator of the information resource in the sense defined in DC, since it denotes "the party that introduced the entity into the KB" - which is not the creator/author of the resource. | **`dc:creator`**: an entity primarily responsible for making the content of the resource. Examples of a Creator include a person, an organization, or a service. Typically, the name of a Creator should be used to indicate the entity. |
| Lacking in a directly corresponding property.<br>The **`protonu:compliantWithSchema`** property partially covers the relation between the **`protonu:Dataset`** and **`protonu:DataSchema`** sub-classes. | **`dc:relation:`** a reference to a related resource. Recommended best practice is to reference the resource by means of a string or number conforming to a formal identification system. |
| The **`protont:InformationResource`** branch of PROTON contains many sub-classes, some of which are quite specific - e.g. **`protonu:MagazineIssue`**. A **`protonu:hasPublisher`** relation (exactly matching the **`dc:publisher`** metadata element in DC in terms of definition and purpose) is provided for the sub-classes that need such a property – in the case with the **`protont:InformationResource`** class branch, these are **`protonu:MagazineIssue,`** **`protonu:MeetingProceedings`**, and **`protonu:NewspaperIssue`**, which inherit the **`protonu:hasPublisher`** property from their other (multiple) super-class **`protont:Document`**. | **`dc:publisher:`** an entity responsible for making the resource available. Examples of a Publisher include a person, an organization, or a service. Typically, the name of a Publisher should be used to indicate the entity. |
| **`protont:partOf`**: the general part-of relation which in place between a whole and each of its parts. It has number of specializations. | **`dc:isPartOf:`** the described resource is a physical or logical part of the referenced resource. Label: *Is Part Of*. Type of term: metadata element refinement. |

| | |
|---|---|
| **`protont:locatedIn`**: it has just a partial correspondence, in special cases, to the **`dc:spatial`** metadata element refinement of DC. | **`dc:spatial:`** the described resource is a physical or logical part of the referenced resource. Label: *Is Part Of*. Type of term: metadata element refinement. |
| **`protont:validFrom`** and **`protont:validUntil`**: these define the range (period) of validity of the information resource. | **`dc:valid`**: date (often a range) of validity of a resource. Type of term: metadata element refinement. |

A mapping between the subclasses of **`protont:InformationResource`**, including the **`protont:Document`** sub-class branch, and **Dublin Core**, follow:

**`protonu:DataSchema`**:
A particular notation for representation, standardization, and/or structuring of information. It can range from db schema, through ontology, to any sort of taxonomies, nomenclatures and subject hierarchies. Examples are Dublin Core, KIMO, SIC, XML, RDFS. DC-relevance: **full equivalence** between the **`protonu:DataSchema`** sub-class in PROTON and the **`dc:title`** metadata element (labelled *Title*) in DC.

**`protonu:Dataset`**:
A dataset is information encoded in a defined structure (for example, lists, tables, and databases), intended to be useful for direct machine processing (DCMI Type). Somehow structured and interrelated body of information, data, or knowledge. This includes databases, knowledge bases, catalogues, registries, specific lists, etc. All tangible aspects (like media or host) are irrelevant for this class - it only considers the abstract information. What can be considered as a single document is outside the scope of this class, although it can be comprehensive in terms of size and structure.

**`protonu:Legislation`**:
Various sorts of legislative documents, including constitutions, laws, etc.

**`protonu:Patent`**:
A registered (or awaiting registration) patent for a specific invention or design.

**`protonu:ResourceCollection`**:
A collection is an aggregation of information resources. The term collection means that the resource is described as a group; its parts may be separately described and navigated. (DCMI type Collection) DC-relevance: here the **`protonu:ResourceCollection`** sub-class in PROTON is aligned to the much more broadly defined **`dc:collection`** metadata element (labelled *Collection*) in DC, which is described as "an aggregation of items" – i.e. the type of "items" is not particularized to more concrete terms or "entities".

**`protonu:MagazineIssue`**$^{M}$:

A specific issue of a magazine, journal or digest. Being a multiple-inheritance class[15], **protonu:MagazineIssue**$^{M}$ has two (multiple) super-classes – the other super-class, **protonu:IssueOfPeriodical** (and its **protonu:PublishedMaterial** super-class), define the additional properties, inherited by **protonu:MagazineIssue**$^{M}$ – **protonu:issueOf**, **protonu:datePublished**, and **protonu:hasPublisher**.

**protonu:MeetingProceedings**$^{M}$:
A collection of articles or presentations published as a book. Being a multiple-inheritance class[15], **protonu: MeetingProceedings**$^{M}$ has two (multiple) super-classes – the other super-class - **protonu:Book** - defines the additional property **protonu:ISBN** (International Standard Book Number - a unique machine-readable identification number, which marks any book unmistakably; this property aligns well with the **dc:Identifier** metadata element as defined in DC), inherited by **protonu:MagazineIssue**$^{M}$.

**protonu:NewspaperIssue**$^{M}$:
A specific issue of a newspaper. Being a multiple-inheritance class[15], **protonu:NewspaperIssue**$^{M}$ has two (multiple) super-classes – the other super-class, **protonu:IssueOfPeriodical** (and its **protonu:PublishedMaterial** super-class), define the additional properties, inherited by **protonu:NewspaperIssue**$^{M}$ – **protonu:issueOf**, **protonu:datePublished**, and **protonu:hasPublisher**.

**protont:Document** (a PROTON Top module super-class):
The information content of any sort of a document. Any tangible (material) aspects of a document are ignored. It is usually a document in free text with no formal structure or semantics. A distinctive type of an information resource; what distinguishes the **protont:Document** class, as a specific type of an information resource, are the aspects of intention and particularism of a document, at least in the “general” connotations and understanding about “what a document is" and what cannot be described as a document (e.g. a database is not a document, it is rather a dataset).
Proprietary properties/attributes/relations in PROTON:

- **protont:documentAbstract**. PROTON description: An abstract or summary of a document. **protont:documentAbstract** is a direct specialization of the system property **protons:Desription** DC-relevance: this property matches almost in full the Dublin Core metadata element refinement **dc:Abstract**, which is described as “a summary of the content of the resource”.
- **protont:documentSubTitle**. PROTON description: A very short sub-title of a document, usually a single sentence. **protont:documentSubTitle** is a direct specialization of the system property **protons:laconicDesription**., which in its turn is a specialization of **protons:Desription**. DC-relevance: this property does not have a corresponding element in DC. However, it can be regarded as a further specialization of **protonu:Title** – as defined in both PROTON and DC.

**protonu:Contract**:

[15] The superscript $^{M}$ sign in the name of a class in PROTON means that this class is a multiple-inheritance class, i.e. it has two or more superclasses.

Any sort of contract or treaty, as well as other documents signed or otherwise accepted by more than one agent, in the sense of a binding agreement between two or more persons that is, typically, enforceable by law.

**protonu:Message**:
A written message, including various postings in newspapers or public sources, job position adverts, etc. The emphasis here, in this specialization of the term in PROTON, is that this is a written type of a document. I.e. some other meanings of "message" should be ignored.

**protonu:PublishedMaterial**$^{M}$:
A document which is published or intended for publishing. Being a multiple-inheritance class[15], **protonu:PublishedMaterial**$^{M}$ has two super-classes - **protont:Document** and **protonu:MediaProduct**. In this context, publishing is used in all of its main meanings: to prepare and issue for public distribution or sale; to put into print; and to have (one's written work) issued for publication (WorldNet 2.0 - see 8.2). DC relevance: making a resource available (what DC means by "publishing") is not inclusive, comprehensive, and particular enough for the purpose of PROTON.

**protonu:Announcement**:
A formal public statement; "the government made an announcement about changes in the drug war"; "a declaration of independence" (Wordnet, see section 8.2).

**protonu:Article**:
A relatively short document, published as a part of a **protonu:ResourceCollection**. **protonu:Article** has a proprietary property – **protonu:publishedWithin**, which further specializes the class relations to other classes. In a way, this class falls under the broader interpretation of **dc:Text** in Dublin Core: "a text is a resource whose content is primarily words for reading."

**protonu:Book**:
A relatively big published document (as opposed to **protonu:Article**). It may or may not have chapters. There could be a series of books considered as volumes of a bigger one. In a way, this class falls under the broader interpretation of **dc:Text** in Dublin Core: "a text is a resource whose content is primarily words for reading."

**protonu:MeetingProceedings**$^{M}$:
See the description of this multiple-inheritance sub-class above in this section.

**protonu:IssueOfPeriodical**:
A specific issue, number, and/or volume of a periodical publication such as a magazine. The proprietary **protonu:issueOf** property of this sub-class provides a relation to a periodical publication (as it is a property of multiple classes). In this way, a concrete issue of a periodical is matched to the periodical publication in general. The issuance is supposed to be a formal one, as implied in Dublin Core (**dc:Issued** element refinement).

**protonu:MagazineIssue**$^{M}$:
See the description of this multiple-inheritance sub-class above in this section.

**protonu:NewspaperIssue**$^M$:
See the description of this multiple-inheritance sub-class above in this section.

**protonu:Report**:
A report could be a written document describing the findings of some individual or group, or a short account of the news, or, in some specific contexts, it could be assumed the act of informing by verbal report.

### *6.1.7 Product class*

**protont:Product** (a PROTON Top module class, a sub-class of **protont:Object**) covers the general concept of a product model, say, Ford T. The instances of this class are not specific instances of the product - the latter are just objects. Analogous to *FormalProductType* in Cyc (section 8.4). Apart from the properties, inherited from **protont:Object**, **protont:Product** has the proprietary one **protont:producedBy** - a relation between the **protont:Product** and **protont:Company**/**protont:Agent** which produced/produces it. A new sub-class of **protont:Product** is PROTON Knowledge Management module (section 7) class **protonkm:Device** (section 7.9).

### *6.1.8 Service class*

**protont:Service** (a PROTON Top module class, a sub-class of **protont:Object**) denotes any sort of a service, ranging from scheduled flight or train services to weather forecast information/web service. Many services could be considered instances of **protont:Agent**. Apart from the properties, inherited from **protont:Object**, **protont:Service** has the proprietary one **protont:operatedBy** - a relation between a **protont:Service** and a **protont:Agent** (usually an organization), which provides it.

### *6.1.9 CommercialOrganization class*

**protonu:CommercialOrganization** (a PROTON Upper module class, a sub-class of **protont:Organization** and a super-class of **protonu:Company**) denotes an organization, which buys or sells goods or services for a profit. It may also be a Business or it may merely be a sub-organization of a Business entity.

The reason for the existence of **protonu:CommercialOrganization** and the further specialization of **protonu:Company** as its sub-class is the equivocality of a potential assumption that "every commercial organization is a company, so we do not need both". Actually, a **protonu:Company** denotes entities that are subject to more specific (even constraining to some extent) conditions compared to its superclass, i.e. its instances are *private, legal*, *corporate* entities with the *legal rights to own property, manage themselves, and sue or be sued*; companies are established by a charter or registration granted by a government.

Apart from the properties, inherited from **protont:Object**, **protonu:CommercialOrganization** has several proprietary ones, as follows:

- **protonu:FISCAL_NET_INCOME** - net income during the last fiscal year;
- **protonu:FISCAL_SALES** – sales during the last fiscal year;
- **protonu:activeInSector** - denotes that the organization is active within the respective industry sector. This property relates

**protonu:ComercialOrganization** with **protonu:IndustrySector** (a sub-class of **protonu:BusinessAbstraction**, sections 6.3.6 and 6.3.5);

- **protonu:hasShareholder** - relates a particular organization to the agents, which are members of that organization. This predicate indicates a `generic' membership, although there may be specialized kinds of membership in the same organization. Typically, membership eligibility is determined by the organization and it is accepted with the agent's voluntary affiliation. In many cases, **Person**s who take **Position**s within an **Organization** are considered members of that organization, although this is not formally encoded here in any way.

## 6.2 Happening branch

**protont:Happening** (a PROTON Top module class) is something that happens. It can be either dynamic – like in "drawing a circle" (the **protont:Event** sub-class), or static – like in "being a president" or “sitting on a chair” (the **Situation** sub-class). In all cases, a happening has a certain (usually quite concrete) temporal positioning (extent) – in the simplest case one, denoted by start and end points in time. Its proprietary relations and attributes are: **protont:startTime** (the starting moment of a happening) and **protont:endTime** (the end point of a happening).

Also, it is usually the case that some entities[16] take part in the happening – i.e. are involved in, or play a certain role in it. To this end, PROTON contains the **protont:Role** sub-class (see 6.2.5).

The **protont:Happening** class is a super-class of the following PROTON Top module classes: **protont:Event** (section 6.2.1), **protont:Situation** (section 6.2.2), **protont:TimeInterval** (section 6.2.3), and **protont:JobPosition** (section 6.2.4).

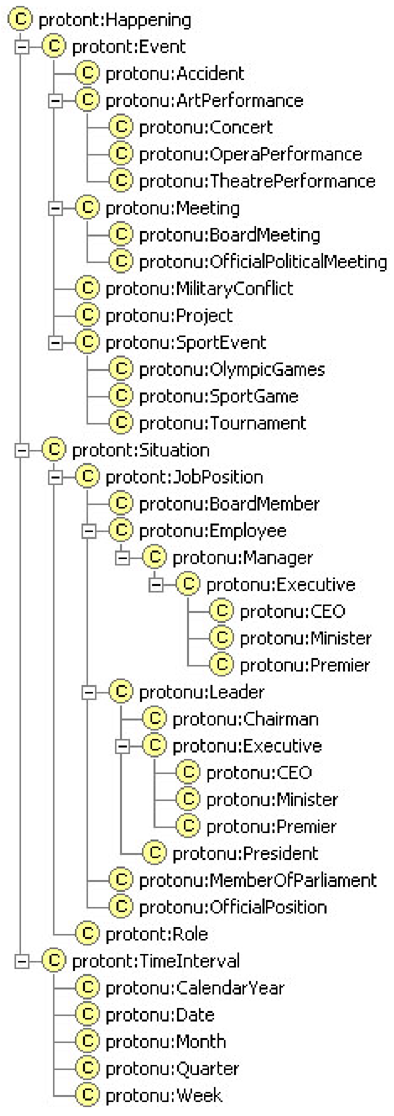


**Fig. 13. The PROTON Top module class Happening and its Upper module branch**

[16] This is a rather unrestricted modelling; if we have to be more specific, only Objects and Abstract entities are expected to take part in happenings.

### 6.2.1 Event class

**`protont:Event`** (a PROTON Top module class) denotes a dynamic event, such as "running", or "a concert". **`protont:Event`** is in binary logical opposition to **`protont:Situation`** in terms of the dynamic vs. static nature of the **`protont:Happening`**. **`protont:Event`** inherits the properties of its super-class **`protont:Happening`**. Its specializations include sub-classes like **`protonu:Accident`**, **`protonu:Meeting`** (section 6.2.6), **`protonu:Project`** (6.2.7), **`protonu:SportEvent`**, **`protonu:MilitaryConflict`**, and **`protonu:ArtPerformance`**.

### 6.2.2 Situation class

**`protont:Situation`** (a PROTON Top module class) denotes a static event or situation, like "sitting on a chair" or "holding position". Typically, those are temporarily homogeneous, i.e. their nature is not expected/required to change within their duration. As a **`protont:Happening`**, they use to happen/take place/be true for some periods of time and may or may not have a well-defined space extension. **`protont:Situation`** is in binary logical opposition to **`protont:Event`** in terms of the dynamic vs. static nature of the **`protont:Happening`**. **`protont:Situation`** inherits the properties of its super-class **`protont:Happening`**. Its specializations include sub-classes like **`protont:JobPosition`** (section 6.2.4) and **`protont:Role`** (6.2.5).

### 6.2.3 TimeInterval class, modelling of time

**`protont:TimeInterval`** (a PROTON Top module class) is a general time expression (TIMEEX), which refers to a particular period of time, an interval. Repeating periods (like *the Spring* or *Christmas*) are not time intervals, while specific instances of theirs (like *the Spring of 1944*) are. A **`protont:TimeInterval`** could collapse, in very special cases, to a time point, however in this case, in contrast to the **`protont:Abstract`** time point (referring to some time during the day), it should be bound to a specific date, i.e. to represent a timestamp.

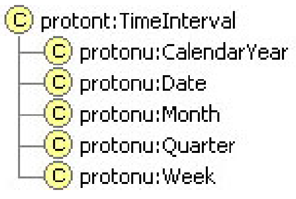


**Fig. 14. protont:TimeInterval class hierarchy**

**`protont:TimeInterval`** in PROTON is considered a specialization of **`protont:Happening`**, because this is the place in PROTON for all entities that are largely defined through their *temporal extent*. Indeed, it is not a necessity that something should be “happening” (in the primal literal sense of the word) during a time interval, but on the other hand we have no doubt that every “happening” surely takes place within a time interval (otherwise it should not be classified as an instance of **`protont:Happening`**). Therefore, in a much broader sense, the sheer demarcation of a specific period of time into a Time Interval could hardly be triggered off simply by chance, without anything “happening” within (and also beyond) this period of time: it is rather vice versa - the start and end points in time mark in and out the beginning and the closing, in-between, or final point of an event, a process, an accident, or something that takes (is taking) place and for some reason its happening is important in the global context of events. To go further, a “happening” is not *nolens volens* anything tangible, or dynamically discernible, or appreciable by human senses, or consciously noticeable, or substantially perceptible, or necessarily visible - it is

Kantian subjective reality[17] that gives the essence of a time interval as a space of time between events or states, the designation of the latter two as such being highly a matter of fairly subjective interpretation.

Abstract seasons, months, days of the week are represented as sub-classes of the **`protonu:TemporalAbstraction`** in the **`protont:Abstract`** branch of PROTON. Thus, the general notions of Tuesday and July appear as instances of classes in this branch. It ought to be made clear that those do not have a temporal extent – they are just abstractions, symbols, which can be used to refer to particular periods of time.

### *6.2.4 JobPosition class and modelling*

**`protont:JobPosition`**-s (a PROTON Top module class) of people within organizations are modelled as a special sort of a situation, i.e. a "static" happening. This matches their nature well, as seen in the following context: "it happened that he was a CTO at XYZ Corp between 2001 and 2003." Its proprietary relations and attributes are:

- **`protont:holder`** – relates the position to the **`protont:Person`** who holds it. There is an inverse relation to this one, named **`hasPosition`**, defined for persons.
- **`withinOgranization`** – the **`protont:Organization`** where the position is situated.
- **`protont:heldFrom`** and **`protont:heldTo`** – two literal-ranged attributes, to be used to encode the start and end of the period, for which the particular person used to take this position, if known. Those are specializations respectively of the **`protont:startTime`** and **`protont:endTime`** attributes of **`protont:Happening`**.

There is a more specialized support for official (e.g. government positions), through the **`protonu:OfficialPosition`** sub-class with an **`protonu:officialPositionIn`** property ranged **`protont:Location`**. The latter provides a useful shortcut as seen on Fig. 15. (the dashed arrows and nodes.)

[17] I.e. the concept that ideas can have truth independently from the reality outside of our minds, and that reality is only known when it conforms to the mind that holds its knowledge (Kant, I. *The Critique of Pure Reason*). According to Kant, all that is stretched beyond our experience is unknowable, but at the same time it is necessary to presume that those things beyond our perception exist – they are real despite that they are not subjectively essential. Along this line of thoughts, an event can be deemed even a strip of silence during a concert perofmance of an orchestra: now, does silence "happen" or not? Or is it a state?

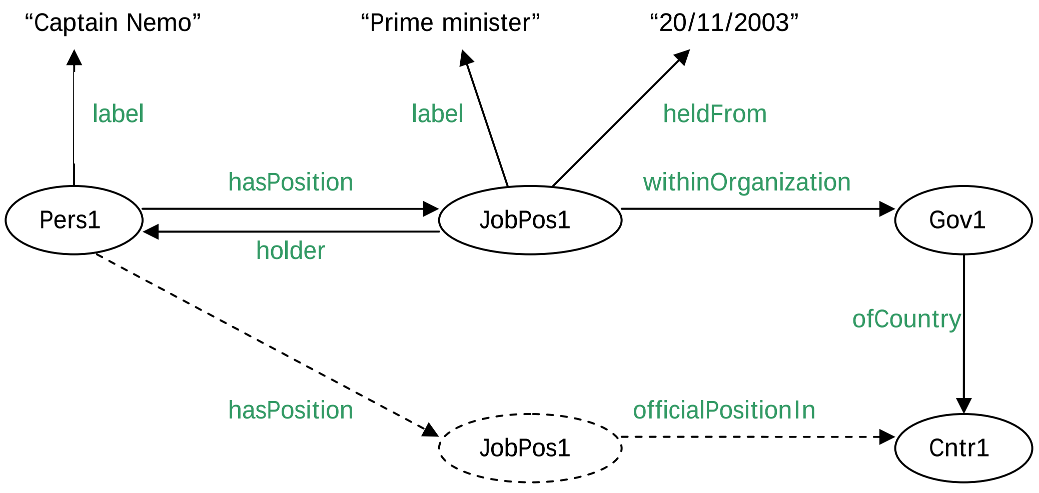


**Fig. 15. Modelling of (Official) JobPosition-s**

There is a general temptation (or expectation) to model **`protont:JobPosition`**-s with relationships (instead of classes/instances), e.g. **`<pers1, pos1, org1>`**. It would work well if those were pure binary relations, but, as it can be seen at Fig. 15. there is more information (than just source and target) to be managed. Then, an alternative to the current approach would be a sort of an n-ary relation or reification, none of which could be judged as a better option.When it comes to using sub-classes of the **`protont:JobPosition`**, it should be taken into account that such a sub-class should still be instantiated to model a particular position. Fig. 16. below gives an example how the **`protonu:Executive`** sub-class of **`protont:JobPosition`** in PROTON can be used.

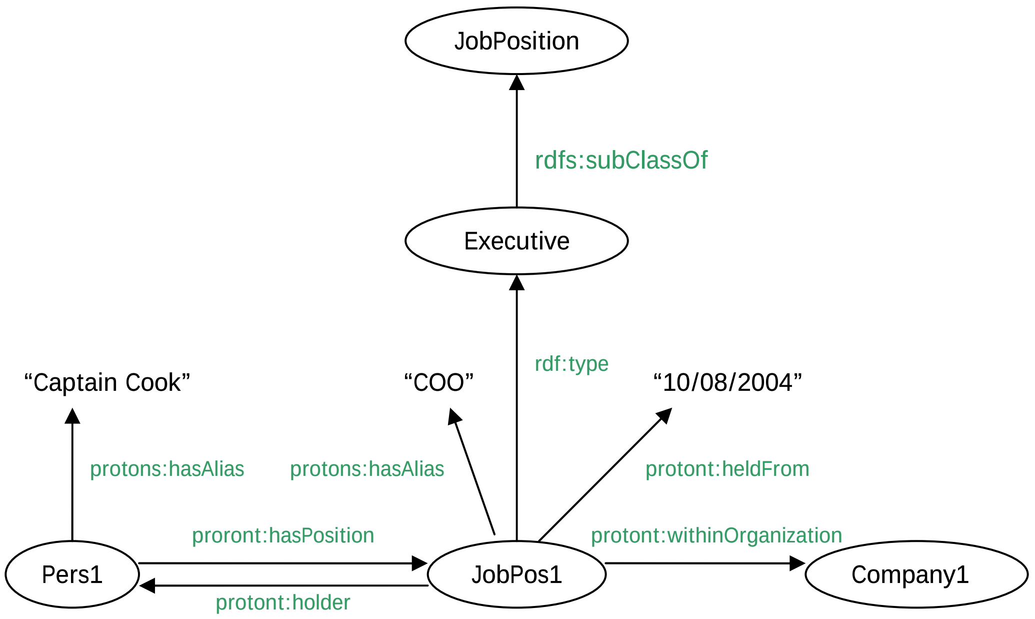


**Fig. 16. Modelling of a sub-class of protont:JobPosition**

### *6.2.5 Role class*

A **`protont:Role`** (a PROTON Top module class) designates the role of an entity (usually an agent) within/for/during/effecting (intentionally or not) a particular happening. For instance, the role played by a coordinator of a project, or a defendant in a trial, or even an evidence in a trial (e.g. a material object that serves as an evidence, for example a knife a murder was committed with). Its proprietary properties are:

- **`protont:roleHolder`** – relates the role to the **`protons:Entity`** that holds ("plays") it.
- **`protont:roleIn`** – relates the role to the **`protont:Happening`** that conditions and takes on the effect of the role.
- **`protonkm:hasRole`** – relates the role to the **`protonkm:UserProfile`** (section 7.6) to link the **`protonkm:User`** (section 7.3) with the role he/she plays with respect to a certain system(s).

As shown in the example in Fig. 17. when sub-classes of **`protont:Role`** are to be modelled (e.g. in a domain ontology), it should be taken into account that interdependencies between some roles are highly possible, so that transitive role-to-role relations might be necessary at some point.

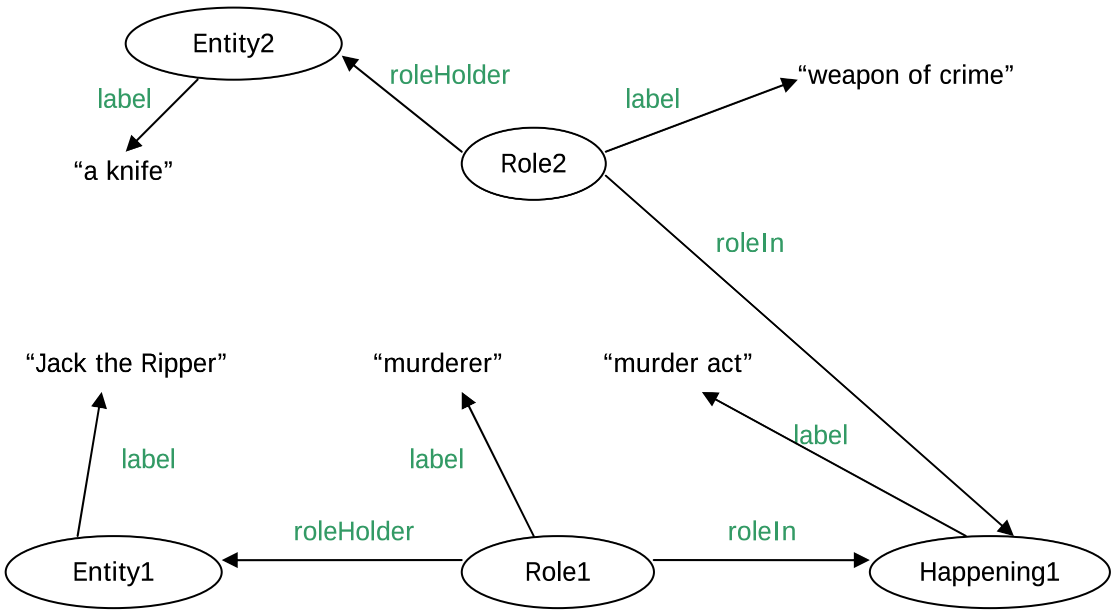


**Fig. 17. Modelling of the protont:Role class**

Moreover, by an axiom in PROTON, when the **`protont:roleHolder`** of a role instance is an instance of a **`protont:Agent`**, then this **`protont:Agent`** is assumed to be **`protont:involvedIn`** the respective **`protont:Happening`**.

### *6.2.6 Meeting class*

**protonu:Meeting** (a PROTON Upper module class, a sub-class of **protont:Event** – section 6.2.1) denotes a formal or informal meeting, regarded as an event (i.e. it is specific in temporal and spatial aspects). It is worth noting that **protonu:Meeting** here primarily implies the following senses of the word – i.e. a coming together of persons or things, or an assembly, gathering of people, especially to discuss or decide on matters. **protonu:Meeting** has no properties of its own - it inherits the ones of its super-class.

### *6.2.7 Project class*

**protonu:Project** (a PROTON Upper module class, a sub-class of **protont:Event** – section 6.2.1) may denote a special unit of work, research, etc.; a proposal of something to be done, like a plan or a scheme; an organized undertaking; an extensive public undertaking, as in construction, conservation, etc. In any case, a **protonu:Project** has a duration in time, i.e. it has its temporal connotations. **protonu:Project** has no properties of its own - it inherits the ones of its super-class.

### *6.2.8 OfficialPosition class*

**protonu:OfficialPosition** (a PROTON Upper module class, a sub-class of **protont:JobPosition** - section 6.2.4) denotes a specific position, which models the **Person**->**hasPosition**->**Location**, e.g. "president of USA", "mayor of NY". It is usually a shortcut to holding a position within the local government or other local administration, including military positions. Apart from the properties it inherits from its super-class - **protont:JobPosition**, **protonu:OfficialPosition** has the proprietary property **protonu:officialPositionIn**, which relates it to the **protont:Location** class (section 6.1.4).

### *6.2.9 Employee class*

**protonu:Employee** (a PROTON Upper module class, a sub-class of **protont:JobPosition** - section 6.2.4) denotes an employee of a **protont:Group** (usually a **protont:Organization**). It has **protonu:Manager** as a sub-class, and on the next level, the sub-class **protonu:Executive**$^{M}$ [18] (a **protont:Person** who holds an executive managerial **protont:JobPosition** in a certain **protont:Organization** ; **protonu:Executive-**s are top managers of organizations, including corporate officers (CompanyPresident, etc.), Chiefs of Staff, Generals, Admirals, and others like Chief Corporate Counsel, Managing Partner, Producer, Chief Scientist, Chief Engineer, as well as other top and middle-top managers; at the same time, a **protonu:Executive** is usually employed by the owners of the respective Organization), which is a multiple-inheritance sub-class of both **protonu:Manager** and **protonu:Leader**. **protonu:Employee** has no properties of its own - it inherits the ones of its super-class.

### *6.2.10 Leader class*

**protonu:Leader** (a PROTON Upper module class, a sub-class of **protont:JobPosition** – section 6.2.4) denotes a **protont:JobPosition** for a

[18] The superscript $^{M}$ sign in the name of a class in PROTON means that this class is a multiple-inheritance class, i.e. it has two or more super-classes.

**protont:Person** who heads a **protont:Organization**. Typically, a leader of an organization makes major decisions on behalf of the whole organization, has the authority to direct the personnel of the organization to carry out those decisions, and is empowered to engage or negotiate with external agents to achieve the goals of the organization. This collection includes leaders of divisions, such as department heads within larger organizations. Also, a single person may be a leader in more than one organization.

**protonu:Leader** has as sub-classes **protonu:Chairman**, **protonu:President**, and the multiple-inheritance class **protonu:Executive**$^{M}$ (see section 6.2.9). **protonu:Leader** inherits the properties of its super-class **protont:JobPosition**.

### *6.2.11 Date class*

**protonu:Date** (a PROTON Upper module class, a sub-class of **protont:TimeInterval** – section 6.2.1) denotes a specific date, as 12th of April, 1956, as the time period (the 24 hours of the day). **protonu:Date** inherits the properties of **protont:TimeInterval**.

## 6.3 Abstract branch

**protont:Abstract** (a PROTON Top module class) denotes an abstraction: i.e. something, which neither happens nor exists, e.g. a number or a chemical compound. Those are usually some symbols invented to refer to general notions.

The **protont:Abstract** class is a super-class of the following PROTON Top module classes: **protont:Number** (6.3.1), **protont:ContactInformation** (6.3.2), **protont:Language** (6.3.3), and **protont:Topic** (6.3.4).

In some cases, when a class has specific, well-definable instances, which however may not be modelled as sub-classes, those instances are included in the knowledge base (e.g. some special instances of the **protonu:Profession** class - artist, politician, religious person, scientist, sportsman – have been instaniated in the system part of the ontology).

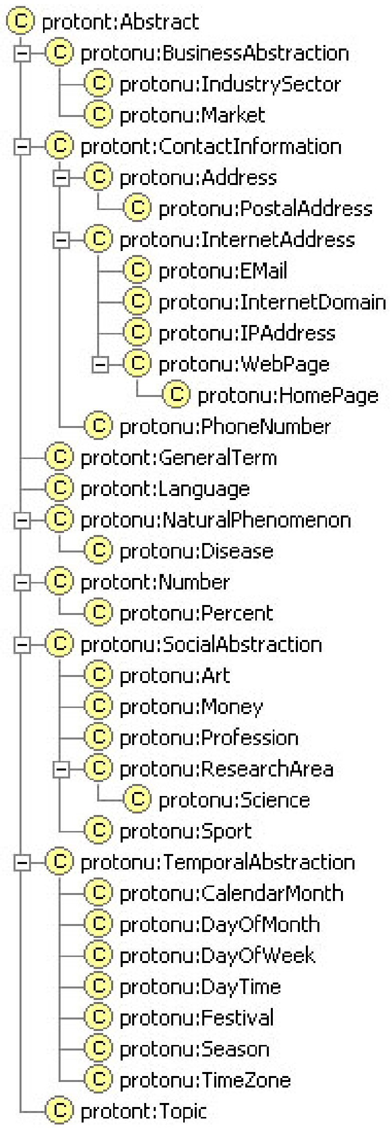


**Fig. 18. protont:Abstract class hierarchy**

### *6.3.1 Number class*

**protont:Number** (a PROTON Top module class) is any given number, within the meaning that a number is: a concept of quantity derived from zero and units (“every number has a unique position in the sequence"); or a number is a numeral or string of numerals used for identification ("she refused to give them her Social Security number"); or a phone number, etc. A further specialization of **protont:Number** is the **protonu:Percent** sub-class, which denotes a specific percent value and thus it is a quite more specific kind of a number in terms of representation and meaning.

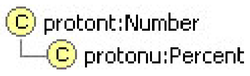


**Fig. 19. protont:Number class**

### 6.3.2 *ContactInformation class*

**`protont:ContactInformation`** (a PROTON Top module class) is any instance of a particular notation, used to make the contact with an individual or an organization possible. The class hierarchy is of the different sorts of contact information is shown on the left-hand side of Fig. 20. (the hierarchy of properties is presented on its right-hand side).

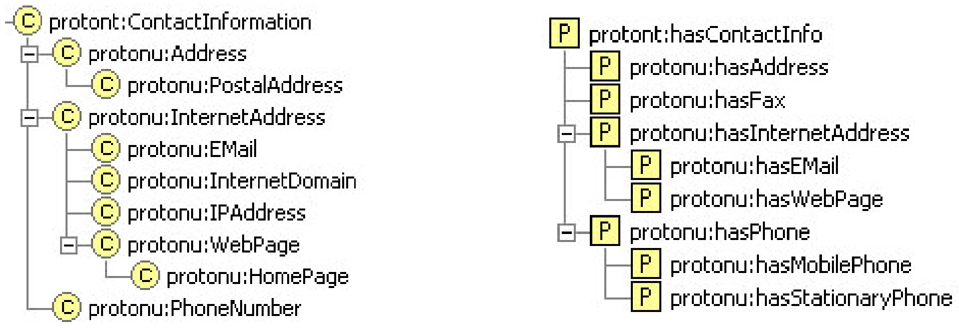


**Fig. 20. Modelling protont:ContactInformation**

### 6.3.3 *Language class*

**`protont:Language`** (a PROTON Top module class) denotes a spoken or written natural language – i.e. a human written or spoken language used by a community; opposed to e.g. a computer language (adapted from Wordnet - section 8.2).

Along with the general properties that the **`protont:Language`** class inherits from the **`protons:Entity`** superclass (defined in the PROTON System module), there is the following important property that concerns **`protont:Language`** but is not proprietary to this class: **`protont:inLanguage`** – it relates the **`protont:InformationResource`** (section 6.1.6) and **`protont:Language`** top module classes. The domain of this property is the **`protont:InformationResource`** class. It links an information resource to the language of the intellectual content of the resource.

### 6.3.4 *Topic class and modelling*

**`protont:Topic`** (a PROTON Top module class) is any sort of a topic or a theme, explicitly defined for classification purposes. As long as any other class or entity can play the role of a topic, the instances of this class are only those concepts, which are defined to serve as topics. The topic class is the natural top-class for linkage of logically informal taxonomies.

PROTON does not provide any **`protont:Topic`** branches as part of its Upper module layer. However, **`protont:Topic`** is in certain relations with some of the classes in the Knowledge Management module of PROTON (section 7):

- **`protonkm:UserProfile`** (section 7.6) and **`protonkm:InformationSpaceProfile`** (section 7.7), where the properties **`protonkm:isInterestedIn`** and **`protonkm:isInterestedIn`** (proprietary for

**protonkm:UserProfile** and **protonkm:InformationSpaceProfile**) relate those two classes with **protont:Topic**;

- **protonkm:WeightedTerm** (section 7.8), where the property **protonkm:hasWeightedTerm** (proprietary for **protont:Topic**) provides a relation between the two classes.

There are many cases when it is useful to have a non-formal hierarchy that does not comply with the formal logic of an ontology like PROTON. In other words, the members in the respective hierarchy branch are such that they cannot be defined as classes, but rather as a mixture of class-like entities and some specific instances of them down the dependency tree. For instance, many big portal sites often provide dynamic, quick-search lists of "entities" to pick from, the relations and entities there being the result of a survey (e.g. words/entities, queried most often) or some particular set of criteria the designer had in mind when organizing such a list. In those cases, it is unacceptable to stick together a class and its instance as a sub-class.

A good example of such a case is available in [14]. We particularly prefer the approach, presented in “Approach 3" there (using a property other than **rdfs:subClassOf** to organize the subject hierarchy), and this is the purpose of the Topic class – it allows the addition of arbitrary pseudo-classes (although formally presented as classes in PROTON), which by convention do not comply with the formal description logic reasoning, used for the rest of the ontology; the aim is to have a "parallel" hierarchy tree under the Topic super-class, containing logically informal taxonomies (e.g. **Travel** > **Continent** > **Africa** >), and to relate its pseudo-classes to the other PROTON classes via a special RDFS property like **rdfs:seeAlso**. Fig. 21. shows this modelling approach. The example figure follows below:

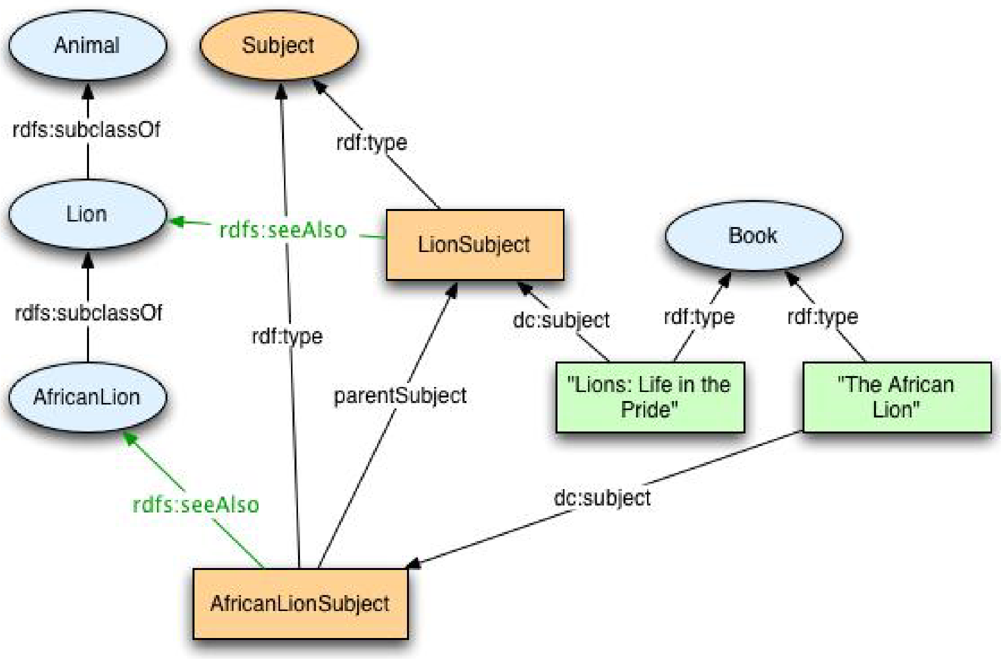


**Fig. 21. Dealing with logically informal classification, using a rdfs:seeAlso property to organize the subject hierarchy (see [14])**

Within PROTON, topics have to be represented as instnaces of the **`protont:Topic`** class (or its sub-classes). **`protont:Topic`**-s can be related to sub-topics via the PROTON **`protont:subTopicOf`** transitive relation - a relation from a less general to a more general topic that is defined to be transitive via a rule. The **`protont:subTopicOf`** property is proprietary for the **`protont:Topic`** class. Typically, the instances of **`protont:Topic`** are values of the **`protont:hasSubject`** property (equivalent to **`dc:subject`**) of the **`protont:InformationResource`** class in PROTON (see section 6.1.6.).

An example for modelling of topics is given on Fig. 22. Suppose one needs to encode that a particular document is about Jazz, using the Yahoo!® category hierarchy. **Jazz**, **Genre**, and **Music** are all instances of **YahooCategory**, which is a sub-class of **`protont:Topic`**.

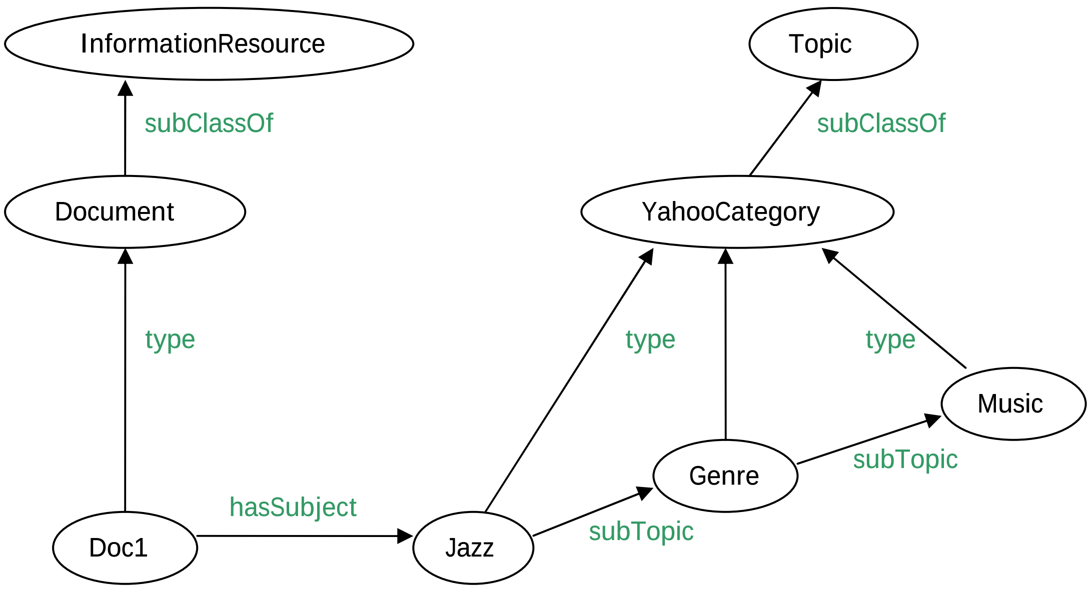
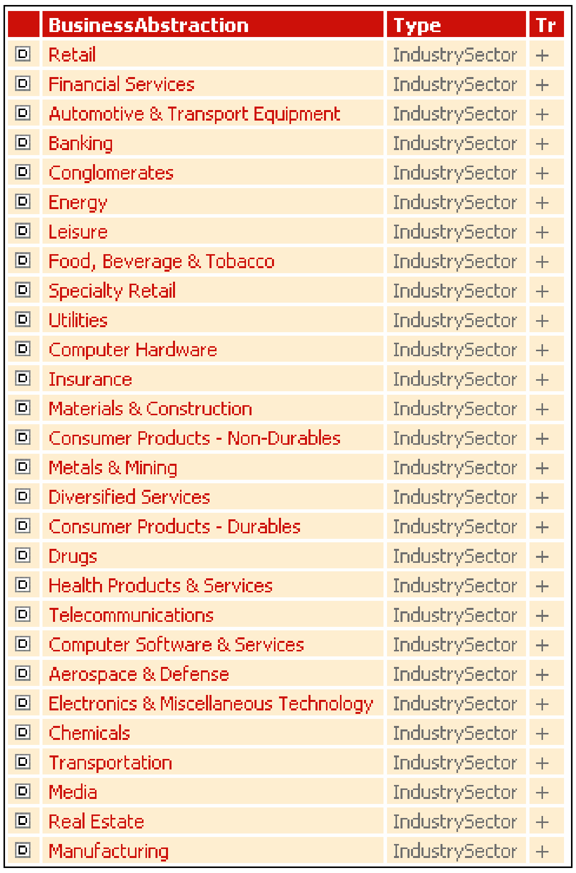


**Fig. 22. Topic modelling example: classifying a document by Yahoo Category**

One can find a more general extension on the distinction between topic- versus schema-ontologies in section 4.3.

### *6.3.5 BusinessAbstraction*

**`protonu:BusinessAbstraction`** (a PROTON Upper module class, a sub-class of **`protont:Abstract`**) denotes an abstract entity that is used in a business context – e.g. markets, industry sectors, brands, etc. Many products can also be seen as a business abstraction, but most of the products bear other important aspects, such as engineering and design.



| | BusinessAbstraction | Type | Tr |
|---|---|---|---|
| ⊡ | Retail | IndustrySector | + |
| ⊡ | Financial Services | IndustrySector | + |
| ⊡ | Automotive & Transport Equipment | IndustrySector | + |
| ⊡ | Banking | IndustrySector | + |
| ⊡ | Conglomerates | IndustrySector | + |
| ⊡ | Energy | IndustrySector | + |
| ⊡ | Leisure | IndustrySector | + |
| ⊡ | Food, Beverage & Tobacco | IndustrySector | + |
| ⊡ | Specialty Retail | IndustrySector | + |
| ⊡ | Utilities | IndustrySector | + |
| ⊡ | Computer Hardware | IndustrySector | + |
| ⊡ | Insurance | IndustrySector | + |
| ⊡ | Materials & Construction | IndustrySector | + |
| ⊡ | Consumer Products - Non-Durables | IndustrySector | + |
| ⊡ | Metals & Mining | IndustrySector | + |
| ⊡ | Diversified Services | IndustrySector | + |
| ⊡ | Consumer Products - Durables | IndustrySector | + |
| ⊡ | Drugs | IndustrySector | + |
| ⊡ | Health Products & Services | IndustrySector | + |
| ⊡ | Telecommunications | IndustrySector | + |
| ⊡ | Computer Software & Services | IndustrySector | + |
| ⊡ | Aerospace & Defense | IndustrySector | + |
| ⊡ | Electronics & Miscellaneous Technology | IndustrySector | + |
| ⊡ | Chemicals | IndustrySector | + |
| ⊡ | Transportation | IndustrySector | + |
| ⊡ | Media | IndustrySector | + |
| ⊡ | Real Estate | IndustrySector | + |
| ⊡ | Manufacturing | IndustrySector | + |

**Fig. 23. An example of a search for entities and documents, relating to protonu:BusinessAbstraction instances, performed with the KIM Web UI front-end**

### 6.3.6 *IndustrySector class and modelling*

Industry sectors are to be represented as instances of the **`protonu:IndustrySector`** class (a PROTON Upper module class), which is a sub-class of **`protonu:BusinessAbstraction`**[19] (section 6.3.5), which on its turn is a sub-class of **`protont:Abstract`** (section 6.3). Hierarchies of industry sectors can be specified with the **`protonu:subSectorOf`** transitive relation (a specialization of **`protont:partOf`**).

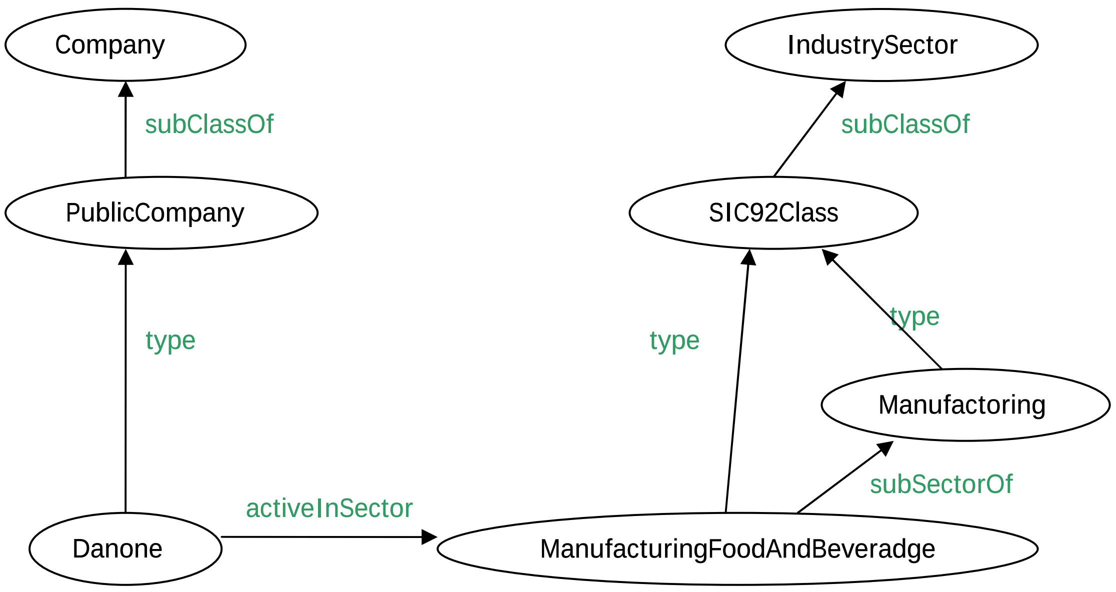


**Fig. 24. Topic modelling example: classifying a document by Yahoo Category**

An example on Fig. 24. demosntrates how the Danone™ Group is modelled as a company, active in the sector of manufacturing food and beverages with respect to the SIC92 classification.

### 6.3.7 *TemporalAbstraction*

A **`protonu:TemporalAbstraction`** (a PROTON Upper module class, a sub-class of **`Abstract`**) denotes any sort of an abstraction that is used to refer to periods of time in general, i.e. periods of time, which are not specific or unique as a concrete date, for instance. Thus, the month of September is an instance of this class, while Sept 1989 is not (it is a specific **`protont:TimeInterval`**, and thus not abstract).

A already hinted, there is a clear, though thin, line of distinction between the two branches in PROTON that deal with temporal aspects of world knowledge – **`protonu:TemporalAbstraction`** and **`protont:TimeInterval`** (section 6.2.3). The key to this delineation lies in the specific or non-specific nature of the temporal notion classified.

[19] This is just a common super-class for any sort of an abstract business entity (such as Market). It has been introduced mostly for the sake of some better structuring of the taxonomy, in order to avoid an excessive number of direct sub-classes of protont:Abstract. See section 6.3.5.

### *6.3.8 GeneralTerm*

A **`protont:GeneralTerm`** (a PROTON Top module class, a sub-class of **`protont:Abstract`**) is the "placeholder" for the representation of a general concept with a well-defined (idiomatic) meaning, which can have a set of distinct lexical items (surface realizations) associated with it. Such examples are: F2F, I18N, P2P, B2B, VIP, ASAP, Semantic Web.

A property, related to the newly integrated layer of PROTON – the knowledge management module (section 7), is the **`protonkm:hasTerm`**, which provides a relation between **`protont:GeneralTerm`** and **`protonkm:WeightedTerm`** (section 7.8). **`protonkm:hasTerm`** is proprietary for **`protonkm:WeightedTerm`**.

### *6.3.9 NaturalPhenomenon*

**`protonu:NaturalPhenomenon`** (a PROTON Upper module class, a sub-class of **`protont:Abstract`**) denotes natural phenomena such as a particular disease, the Gulfstream, or other similar natural abstractions. Important: the particular events or objects, which could instantiate an abstract natural phenomenon (i.e. a specific event of a sickness, caused by a disease) are not instances of this class. **`protonu:NaturalPhenomenon`** inherits the properties of its superclass **`protont:Abstract`**.

### *6.3.10 SocialAbstraction*

**`protonu:SocialAbstraction`** (a PROTON Upper module class, a sub-class of **`protont:Abstract`**) denotes any sort of a general social phenomenon, such as a particular sort of art or science. **`protonu:SocialAbstraction`** inherits the properties of its superclass **`protont:Abstract`**.

# 7 PROTON Knowledge Management Module Classes

The PROTON Knowledge Management (KM) module is dependent on the System and Top modules. Actually, this module is the former SKULO ontology [15], which has been further developed and integrated into PROTON.

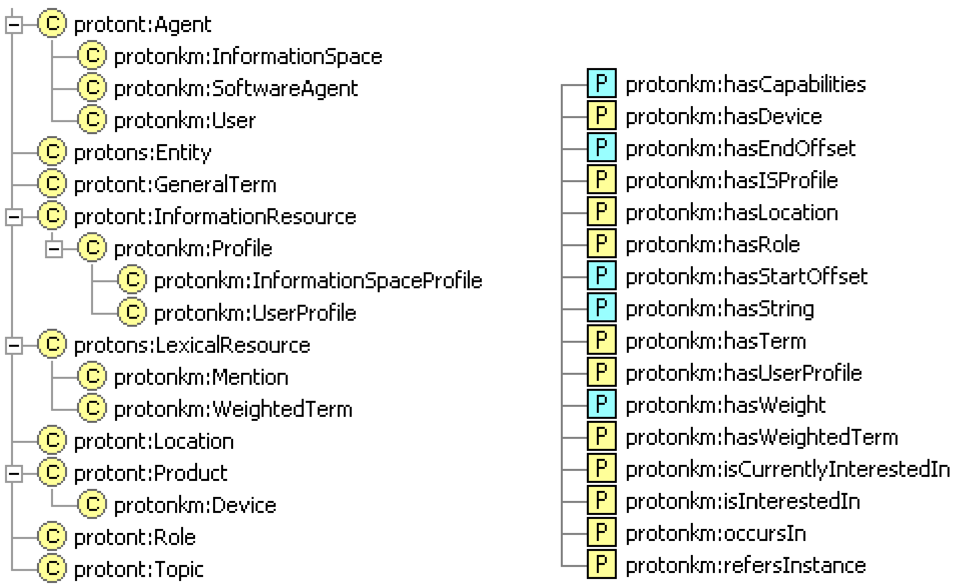


**Fig. 25. PROTON Knowledge Management Module Classes and Properties (incl. imported System and Top modules classes having KM properties as range/domain)**

## 7.1 protonkm:InformationSpace

"Information spaces" denote collections of themed information resources (e.g. documents, maps, etc.) – for instance, the information space "e-commerce" that should contain collections of documents relating to activities and entities concerning electronic commerce. The **`protonkm:InformationSpace`** class is a specialization of **`protont:Agent`** (section 6.1.1), which can be described as denoting a **`protonkm:user`**'s personalized set of information "items" in a specific milieu (e.g. a digital library). Each **`protonkm:InformationSpace`** is linked to an **`protonkm:InformationSpaceProfile`** by means of the property **`protonkm:hasISprofile`**.

## 7.2 protonkm:SoftwareAgent

**`protonkm:SoftwareAgent`** is a specialization of **`protont:Agent`** (section 6.1.1) and denotes an artificial agent, which operates in a software environment. No proprietary properties are associated to this class.

## 7.3 protonkm:User

The concept of a user is central for SEKT, since a key aim of the project is to represent a user's interests and context so that personalised, timely, relevant knowledge is provided [15]. **`protonkm:User`** is a specialization of **`protont:Agent`** (section 6.1.1) and designates a human user, who plays a **`protont:Role`** (section 6.2.5) with respect to some system. Every **`protonkm:User`** has an **`protonkm:UserProfile`** (related via the property **`protonkm:hasUserProfile`**) and this is how the relation between a **`protonkm:User`** and the **`protont:Role`** he/she plays is realized: via the **`protonkm:hasRole`** property that is proprietary for the **`protont:Role`** class.

Each **`protonkm:User`** can have several **`protonkm:UserProfile`**-s, depending on his/her location, device, etc. This means that the **`protonkm:hasUserProfile`** relation is a one-to-many relation.

## 7.4 protonkm:Profile

Every **`protonkm:User`** has a profile, and every **`protonkm:InformationSpace`** has a profile associated with it. The class **`protonkm:Profile`** is a subclass of **`protont:InformationResource`** (section 6.1.6). It has two specializations: **`protonkm:InformationSpaceProfile`** - and **`protonkm:UserProfile`**.

## 7.5 protonkm:InformationSpaceProfile

Each **`protonkm:InformationSpace`** has an **`protonkm:InformationSpaceProfile`**. The relation between them is realized via the property **`protonkm:hasISProfile`** (proprietary for **`protonkm:InformationSpace`**).

An **`protonkm:InformationSpaceProfile`** is typically linked to a set of **`protont:Topic`**-s. The relation between an **`protonkm:InformationSpaceProfile`** and a **`protont:Topic`** is realized by means of the property **`protonkm:isInterestedIn`** (proprietary for **`protonkm:UserProfile`** and **`protonkm:InformationSpaceProfile`**). In addition, each **`protont:Topic`** can have several **`protonkm:WeightedTerm`**-s assigned to it by means of the **`protonkm:hasWeightedTerm`** property (proprietary for **`protont:Topic`**).

## 7.6 protonkm:UserProfile

Each **`protonkm:User`** has an **`protonkm:UserProfile`**. The relation between them is realized via the property **`protonkm:hasUserProfile`** (proprietary for **`protonkm:User`**).

An **`protonkm:UserProfile`** is typically linked to a set of **`protont:Topic`**-s; also, a **`protonkm:UserProfile`** provides information about the location of the user, the device(s) that the user has access to, and the role the user has with respect to a system.

The relation between an **`protonkm:UserProfile`** and a **`protont:Topic`** is realized by means of:

- the property **`protonkm:isInterestedIn`** (proprietary for **`protonkm:UserProfile`** and **`protonkm:InformationSpaceProfile`**) – it is used to relate the user with his/her **long-term** interests in certain topics;

- the property **`protonkm:isCurrentlyInterestedIn`** (proprietary for **`protonkm:UserProfile`**) – it is used to relate the user with his/her **current** interests in certain topics.

Typically, a single **`protonkm:User`** is interested in more than one **`protont:Topic`**, therefore more than one instance of this relation is usually available with a given **`protonkm:UserProfile`** as a domain.

Other proprietary properties for **`protonkm:UserProfile`** are as follows:

- **`protonkm:hasDevice`** - relates **`protonkm:UserProfile`** with the **`protonkm:Device`** class (section 7.9), representing the current device the user has access to;

- **`protonkm:hasLocation`** – relates **`protonkm:UserProfile`** with the **`protont:Location`** class (section 6.1.4), representing the current location of the user;

- **`protonkm:hasRole`** - a user may have one or more roles that they switch between, so this relation links the **`protonkm:UserProfile`** with a **`protont:Role`** (section 6.2.5).

## 7.7 protonkm:Mention

**`protonkm:Mention`** is a specialization of **`protons:LexicalResource`** (section 5.1). Its main purpose is to model the annotations, which are produced by annotation components (provided by WP2 of SEKT) when given a **`protont:Document`** or an **`protont:InformationResource`** [15]. In this context, a **`protonkm:Mention`** represents the mention of an **`protons:Entity`** or a class in an **`protont:InformationResource`** (section 6.1.6). Descriptions of the proprietary properties of **`protonkm:Mention`** follow below.

Attributes (data-properties):

- **`protonkm:hasStartOffset`** – start offset in the content of the information resource;
- **`protonkm:hasEndOffset`** – end offset in the content of the information resource;
- **`protonkm:hasString`** – the string of the annotation.

Relations (object-properties):

- **`protonkm:occursIn`** – relates **`protonkm:Mention`** with **`protont:InformationResource`**;

- **`protonkm:refersInstance`** – relates **`protonkm:Mention`** with **`protons:Entity`**.

## 7.8 protonkm:WeightedTerm

**protonkm:WeightedTerm** is a subclass of **protons:LexicalResource** (section 5.1). It is closely connected to **protont:Topic** - each **Topic** instance may have several **protonkm:WeightedTerm**-s assigned to it. The relation between the two classes is realized via the **protonkm:hasWeightedTerm** property (proprietary for **protont:Topic**). The **protonkm:hasWeightedTerm** relation is a one-to-many relation, i.e. each **protonkm:WeightedTerm** instance is associated with at most one **protont:Topic** instance.

The properties that are proprietary for **protonkm:WeightedTerm** are as follows:

- **protonkm:hasWeight** – a literal property; provides a relation between **protonkm:WeightedTerm** and a real number that expresses the "weight" of the term;
- **protonkm:hasTerm** – relates the with the **protont:GeneralTerm** class (section 6.3.8).

## 7.9 protonkm:Device

The **protonkm:Device** class is a specialization of **protont:Product** (section 6.1.7). A **protonkm:User** can use one or more **protonkm:Device**-s for his/her activities regarding information resource search, management, usage, etc. This relation can be realized via the property **protonkm:hasDevice** (proprietary to the **protonkm:UserProfile** class), which relates the user profile of the user with the device(s) this user works with.

Another property, proprietary for **protonkm:Device**, is the **protonkm:hasCapabilities** relation, which is designed to provide a relation between **protonkm:Device** and a new "Capability" class (still not implemented).

Further details about the attributes and further development issues for the **protonkm:Device** class can be found in [15].

.

# 8 Relations to Other Standards

In the process of development of PROTON, a number of the most prominent upper-level ontologies and metadata standards have been consulted. As a result, some important concepts were directly imported from there – all such cases are noted formally in the built-in documentation of PROTON. Many of the PROTON concepts have been commented informally through descriptions, providing hints to related concepts elsewhere.

## 8.1 Dublin Core

The Dublin Core Metadata Workshop Series began in 1995 with an invitational workshop which brought together librarians, digital library researchers, content experts, and text-markup experts to promote better discovery standards for electronic resources. The Dublin Core (DC) is an up-to-date, authoritative specification of all metadata terms maintained by the Dublin Core Metadata Initiative – elements, element refinements, encoding schemes, and vocabulary terms (the DCMI Type Vocabulary). See [3, 4].

The modelling of documents and other sorts of information resources in PROTON follows Dublin Core, providing direct mapping for all of its elements (see section 6.1.6). Still, some of the names of elements or types are altered for two reasons (i) to match the naming convention of PROTON and (ii) to avoid ambiguity (with notions which are missing in DC, but present PROTON). The alignment is done in two ways:

- Most of the DC elements, [3], are mapped to properties with range **`protont:InformationResource`**;
- Most of the resource types, [4], are represented as sub-classes of **`protont:InformationResource`**

## 8.2 Wordnet

Home page: http://www.cogsci.princeton.edu/~wn/. Online search interface: http://www.cogsci.princeton.edu/cgi-bin/webwn.

Wordnet® is an online lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, adjectives, and adverbs are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets.

While Wordnet is not a true ontology, it is still an extremely valuable upper-level resource which have been consulted through the development of PROTON (and KIMO) and referred in many of the descriptions.

## 8.3 Alexandria Digital Library and GNS

Home page: http://www.alexandria.ucsb.edu/

Alexandria Digital Library (ADL) is a project at the University of California, Santa Barbara. The **`protont:Location`** branch of PROTON contains about 80 classes aligned with the ADL Feature Type Thesaurus [13], which on its turn is aligned with the geographic feature designators, of the GNS database of NIMA (National Imagery and Mapping Agency of United States), at http://earth-info.nga.mil/gns/html/.

## 8.4 OpenCyc

Home page: http://www.opencyc.org.

OpenCyc is a public upper-level ontology, consisting of about 6000 concepts. It is the public part of the Cyc ontology. Cyc, http://www.cyc.com, is probably the biggest and the most prominent AI project on commonsense modelling. The native KR language of Cyc (and OpenCyc) is CycL, which is a relatively expressive one (it compares better to OWL Full and KIF than to the other flavours of OWL or RDFS).

OpenCyc has been consulted during the development of PROTON and it is referred to in many of the descriptions. Here follows a short summary of the differences between OpenCyc and PROTON:

- OpenCyc provides a lot more comprehensive logical definitions of its elements.
- OpenCyc has a much wider coverage. Still, there are elements in PROTON, which are missing in OpenCyc.
- OpenCyc is trying to provide formal modelling of many general notions, such as space and time. It is also the case that PROTON has no extensive coverage of general concepts (such as apple and love), which are not likely to be referred to by name in a text.

In summary, OpenCyc (similarly as DOLCE, [16]) is logically more comprehensive, which makes it suitable for heavy-weight knowledge engineering. However, it also makes the ontology classes more complex and abstract, which means that those are not directly suitable for KM tasks and activities, involving people who are not knowledge engineers.

## 9 Usage and Extension Guidance

Due to its unique modular architecture and its well-organized subsumption hierarchy, PROTON is a flexible, lightweight upper level ontology that is easy to adopt and extend for the purposes of the tools and applications developed within SEKT project. Also, it is quite all-purpose in the sense that it describes very general concepts like space, time, events, objects, abstractions, etc., which for the most part are independent of a particular problem or domain.

In its capacity as a lightweight ontology, PROTON is not packed with an excessive number of logical axioms (unlike comprehensive upper level ontologies) – on the contrary, the lightweight approach used in the process of its development (and the development of its predecessor, KIMO) ensured that it was built with just a basic subsumption hierarchy and a few axioms. As a result, PROTON is a general purpose ontology, it is quite easy to understand and interpret it, and its maintenance, modification, and/or extention would require minor efforts.

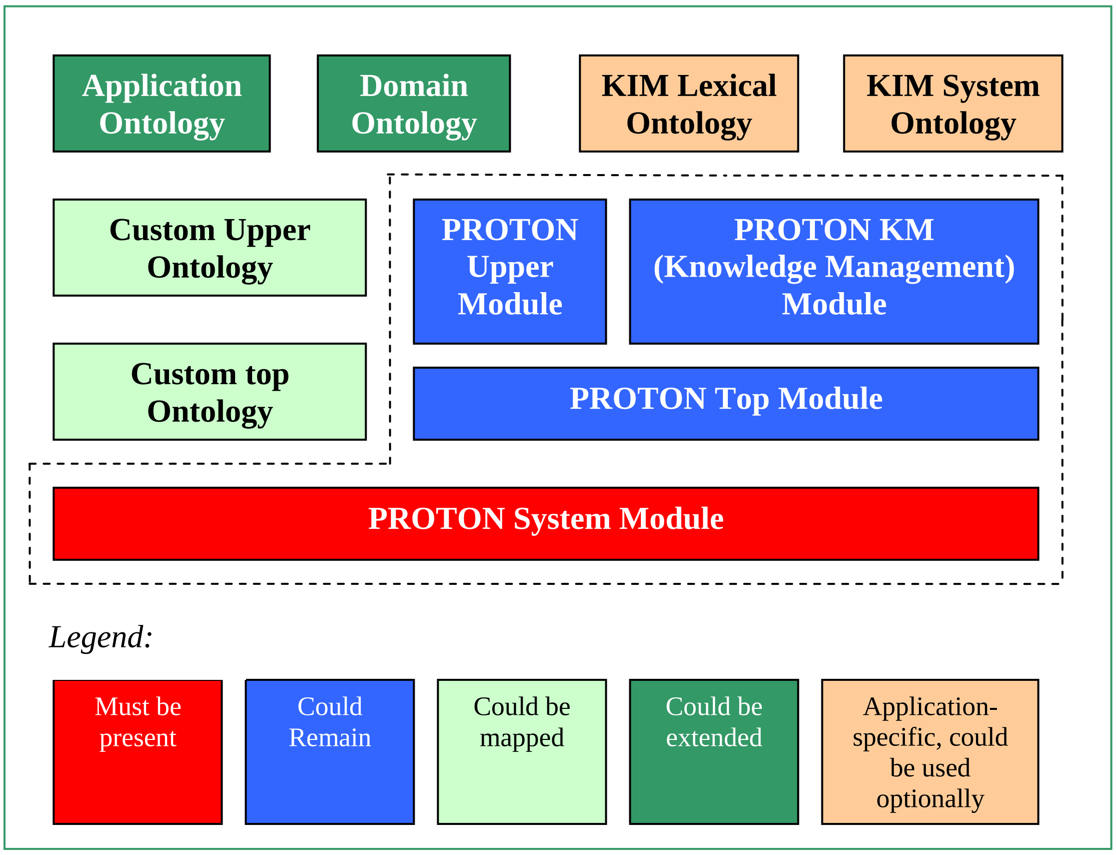


**Fig. 26. PROTON Usage and Extension Guidelines map**

The diagram on Fig. 26. demonstrates the dependencies between the different modules of PROTON and the KIM specific modules. It also depicts potential (possible) extension/customization paths. The dependencies on the diagram are to be interpreted in the direction from the bottom upwards.

The PROTON **System** module is the most basic module. According to expectation, it is to be imported into any ontology or KM tool, developed within the SEKT project.

Further, the **Top** and **Upper** modules or PROTON are subject to choice when it comes to their employment in the development of KM tools and the construction of other ontologies – mainly within SEKT, but also in general. Although we recommend that at least the Top module be used in any development in the scope of SEKT, this is however not a necessity, enforced for the sake of knowledge discovery of metadata generation software. A consideration in this aspect is that the SKULO ontology (now integrated as the PROTON KM module – see section 7) is dependent on the System and Top modules only, i.e. **`protons`** and **`protont`** (e.g. **`protonkm:Device`** is dependent on **`protont:Product`**; **`protonkm:Mention`** is dependent on **`protons:LexicalResource`**; etc.). This means that any software/ontology, which depends on the KM module, [15], should also import the Top module of PROTON along with the System one.

The KIM Platform – as long as its purely software facet is concerned - is dependent on the System module. This means that one can use KIM only via importing the PROTON System module and by extending it with any another domain and/or upper ontology. It should be taken into account that the default information extraction module of KIM is coupled with grammars, which have dependencies on the Top and Upper modules. This means that if the latter are not used, some of the automatic metadata generation capabilities of KIM would vanish, unless the information extraction component gets tuned for the new ontology that would have been put in use.

The most "standard" scenario, as regards the usage and, potentially, the extension of PROTON, should hold that an application, which uses KIM in a specific domain, (i) provides a (proprietary) Domain Ontology, which is an extension of the PROTON Top, Upper, or KM Module, and, (ii) optionally, a Lexical Domain Ontology that is used as a schema for the management of domain-specific lexica (e.g. prefixes and suffixes, which can be used to recognize "unknown" instances of some of the domain classes).

For sure, what has been hard-coded in PROTON, and what therefore stays as a must in the process of integration of PROTON within the process of development of other, more specialized ontologies, applications, and/or KM tools, is the PROTON System Module – the Top, Upper, and KM ones can be well overlooked and substituted, without any "damage", with alternative ones (or simply modified to a certain extent) in case they prove that much unsuitable for the requirements of SEKT-focused tools and applications. However, the System module is quite basic and, at the end of the day, it simply provides a starting place for the mapping of the Top, Upper, and KM modules of PROTON or, potentially, of any additional and/or alternative ontology that may be required by the various connotations of SEKT-specific or general use cases.

Further, a **Custom Top Ontology** and a **Custom Upper Ontology** stand in the places of any other ontologies that may extend or substitute (partially or completely) the proprietary PROTON Top and Upper modules, respectively. Such a replacement is not a hard task, although by expectation the PROTON Top and Upper modules should be efficient, consistent, and generic enough to satisfy the needs of the majority of use cases within the SEKT project, at least. The SKULO ontology (now integrated as PROTON KM module, [15]) can serve as an example of a (more) custom ontology that extends PROTON with those sets of classes, properties, and relations, which are available to all SEKT applications, irrespective of the application domain, but which are not included in PROTON.

**Application Ontology** and **Domain Ontology** are designators of the respective specific application and/or domain ontologies that could be mapped to PROTON as its extensions, according to the requirements of each particular use case.

Lastly, the KIM-specific **KIM System Ontology** and **KIM Lexical Ontology** are mentioned in this diagram due to their smooth integration with the predecessor of PROTON in the past – the KIMO ontology, moreover as this fact allows for their easy use (and reuse) when (and if) that may prove relevant.

Further details on ontology engineering on the basis of PROTON within the scope of the SEKT project can be found in [15].

# 10 Future Development and Community Process

In its capacity as a newborn ontology, PROTON is about to tread the lengthy path of further development and improvement efforts, which should be mainly driven by the community that uses, implements, and extends it, via an apposite community process. Among the number of immediate issues, which are to be taken into account in the near future, the following ones stand out.

## 10.1 Community Process

A community process involving all SEKT partners and workpackages is currently being set up so that the various comments, suggestions, arguments, and issues, relating to PROTON, could be set forth and out and put to discussion in a fairly structured and easily traceable way. In this way such issues could be ultimately resolved for the sake of some consensual ontology improvements.

The results of this process and any consequential updates to PROTON are going to be reflected in this guidance as updates of the latter at certain regular intervals.

A procedure for the technical organization of such a community process could involve the use of tools like Wiki[20] and Plone[21]. Also, it would be interesting to investigate the possible application of the DILIGENT methodology [17] (also available from University of Karlsruhe, Institute AIFB) with the purpose of engineering of upper-level ontologies like PROTON (what is more, such an attempt[22] has already been framed by AIFB).

A currently known issue for discussion within the future community process is the potential further refinement of the modular architecture of PROTON. The modular dividing line between the Top and Upper modules allows for a great extent of flexibility; however, in the process of PROTON usage and extension, this border line may prove slightly imprecise or indistinct. A suitable in-community discussion might lead to considerations that certain Top module classes be moved to the Upper module, and/or vice versa (e.g. the **`protont:hasPosition`** and **`protont:isBossOf`** properties). Further, the Upper module may be broken down into smaller sub-modules (i.e. the **`protont:Location`** branch).

## 10.2 Refinement of protont:Location

A refinement of the properties and sub-classes in the **`protont:Location`** branch (section 6.1.4) appears highly desirable. The need of such an improvement has been triggered by known cases when the current status of some properties may lead to factually untrue inferences, for instance:

- **`protont:subRegionOf`** is an **`owl:TransitiveProperty`**;
- "Diego Garcia" is a **`protonu:MilitaryAreas`** located in the Indian Ocean;
- "Diego Garcia" **`protont:subRegionOf`** "USA";
- "USA" **`protont:subRegionOf`** "North America";

[20] http://wiki.org
[21] http://plone.org
[22] http://www.sekt-project.com/internal/sektwikis/diligent/SektTopLevelOntology

- therefore, because of the transitivity of **`protont:subRegionOf`**, it may be wrongly assumed that "Diego Garcia" **`protont:subRegionOf`** "North America".

### 10.3 Remodelling of protonu:Employee

It is likely a good idea to re-model the current **`protonu:Employee`** class into a property. The reason for such a change may be that the current class status of this position may lead to problems with metadata tools, since the "employee" status of a person should be considered more a *relation* between a **`Person`** and an **`Organization`** (or, in some cases, between two **`Persons`**) rather than an *instance* of an **`Employee`** class.

### 10.4 Slimming down multiple-inheritance classes

The multiple-inheritance of classes at the lower levels should be reduced to an adequate minimum in order to avoid problematic metadata generation due to potential ambiguity or inconsistent results when processed by metadata tools.

### 10.5 Remodelling of protons:EntitySource and protons:generatedBy

There are two issues to be taken into account as concerns the **`protons:EntitySource`** and **`protons:generatedBy`** conceptualizations:

- although helpful for the SEKT-related applications, projects, and tools, the current modelling approach narrows the use of this additional information to instances only. For instance, additional information on concepts/properties might be of interest for SEKT WP1/WP3; or, information on property extensions might sometimes prove essential for SEKT WP2;
- the **`protons:Recognized`** and **`protons:Trusted`** subclasses may be considered to be remodelled as **`owl:AnnotationProperty`**, since their semantics may appear not clear enough and even arbitrary to a part of the community.

### 10.6 Remodelling of other System module entities

The state of affairs, described in the previous section, is analogous for **`protons:LexicalResource`**, **`protons:Alias`**, **`protons:hasAlias`**, **`protons:hasMainAlias`**, **`protons:description`**, and **`protons:laconicDescription`**. In a similar fashion, although such conceptualizations are generally quite useful for SEKT, there is a serious need for a possibility to encode rich lexical information and descriptions not only for instances, but also for concepts, properties, and even property extensions. A possible solution to this issue could be the remodelling as **`owl:AnnotationProperty`**.

### 10.7 Remodelling of protonu:PublicCompany

The **`protonu:PublicCompany`** (a sub-class of **`protonu:Company`**) is defined as "a company, which is publicly traded on some stock exchange". Firstly, such a specialization perhaps goes too far into detail, considering that PROTON is an upper-level ontology. Secondly, metadata generation tools based on IE typically experience problems with multiple classifications of instances – i.e. in our case, there would be a somewhat artificial dividing line between private companies and ones that may be private or not, but which are publicly listed on a stock exchange. Such a specialization seems quite domain-dependent and this feature would better be modelled as a proprietary property of **`protonu:Company`** instead of a sub-class.

### 10.8 Conversion of protonu:hasUniversity

The **`protonu:hasUniversity`** property of **`protont:Location`** does not seem to be the best choice for a property that indicates the existence of relations between organizations and locations. Therefore, a more generic property could be worked out to this end.

# 11 Conclusion

This document constitutes a guidance to the PROTON ontology (PROTo ONtology), developed within the scope of the SEKT project as a basic upper-level ontology to be used as background or pre-existing knowledge schemata for the purposes of metadata generation (WP2) and also as a groundwork for the overall knowledge modelling and integration strategy of a KM environment. In its capacity as a guidance material, this paper is complementary to the PROTON ontology itself, which provides its own internal documentation. In addition to the latter, this document presents design rationales and decisions, comments on the alignment to other ontologies and metadata schemata, and an introductory exposition of the major layers and branches in PROTON.

The main ingredients in the paper are the extensive descriptions of the four modules of PROTON – System, Top, Upper, and KM, including detailed elaboration on all the Top module classes and also some of the Upper module classes and branches that bear more relevance to SEKT-related issues.

As PROTON is a natural successor of the KIMO ontology (created and used for the purposes of the KIM Platform), the paper discusses the main points relating to the further development of PROTON, the relations and dependencies between the two ontologies, as well as the design principles observed in the process of designing the modular architecture of PROTON. In addition, the paper provides a general introduction to ontologies, knowledge representation, and ontology languages, as well as a discussion on the scope, coverage, and compliance of PROTON and its relations to, and alignment against, other standards (section 7).

The 300 classes and 100 properties in PROTON cover most of the upper-level concepts necessary for semantic annotation, indexing, and retrieval. The smart modular architecture of PROTON, combined with its simple subsumption hierarchy, its independence from KIM-related concepts (relocated to specific, separate modules), its OWL Lite representation, and its SEKT-specific tuning, makes PROTON a flexible, lightweight, upper level ontology that is easy to adopt and extend for the purposes of any ontologies or KM tools and applications. Moreover, PROTON is quite generic in the sense that it describes very basic spatial, temporal, material (“physical”), and abstract concepts of world knowledge, which for the most part are independent of a particular problem or domain.

An usage and extension guidance is hereby presented for PROTON, including a detailed usage and extension guidelines map diagram.

In conclusion, PROTON can be defined as a modular, lightweight, upper-level ontology that has the following major assets:

- domain-independent;
- compliant with popular metadata standards;
- provides light-weight logical definitions;
- ensures an broad coverage of concrete and/or named entities;
- requires minimal support for general concepts, which ensures the easy extension in this direction;
- encoded in OWL Lite;

- contains a minimal set of custom entilement rules (axioms).

We want to thank to Denny Vrandecic who dedicated considerable time and energy discussing with us PROTON itself and the organisation of the community process. PROTON received a number of important improvement as a result of the discussions related to the case studies of SEKT and their "ontological" needs – we want to thank for this to all the partners, but most specially to Pompeu Casanovas and Nuria Casellas Caralt (who motivated as to write down and Design Rationale sections) and to John Davis and the BT team (for donating the KM module of PROTON).

## 12 References

1. Brickley, D; Guha, R.V, eds. *Resource Description Framework (RDF) Schemas*, W3C http://www.w3.org/TR/2000/CR-rdf-schema-20000327/
2. Chinchor, N.; Robinson, P. *MUC-7 Named Entity Task Definition* (version 3.5). In Proc. of the MUC-7. 1998.
3. DCMI Usage Board. (2003). *DCMI Metadata Terms.* http://dublincore.org/documents/2003/11/19/dcmi-terms/
4. DCMI Usage Board. (2003). *DCMI Type Vocabulary.* http://dublincore.org/documents/2003/11/19/dcmi-type-vocabulary/
5. Dean, M.; Connolly, D.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D.; Patel-Schneider, P.; Stein, L.A. *Web Ontology Language (OWL) Reference Version 1.0.* W3C Working Draft 12 Nov. 2002, http://www.w3.org/TR/2002/WD-owl-ref-20021112/
6. Fensel, D. *Ontology Language, v.2 (Welcome to OIL)*. Deliverable 2, On-To-Knowledge project, Dec 2001. http://www.ontoknowledge.org/downl/del2.pdf
7. Gruber, T. R. *Toward principles for the design of ontologies used for knowledge sharing*. In N. Guarino & R. Poli, (Eds.), International Workshop on Formal Ontology, Padova, Italy, 1993. http://ksl-web.stanford.edu/KSL_Abstracts/KSL-93-04.html
8. Guha, R.; McCool, R. *TAP: Towards a Web of data*. http://tap.stanford.edu.
9. Mahesh, K.; Nirenburg, S.; Cowie, J.; Farwell D. *An Assessment of Cyc for Natural Language Processing*. MCCS Report, New Mexico State University, 1996.
10. Manov, D.; Kiryakov, A.; Popov, B.; Bontcheva, K.; Maynard, D.; Cunningham, H. *Experiments with geographic knowledge for information extraction*. NAACL-HLT 2003, Canada. Workshop on the Analysis of Geographic References, May 31 2003, Edmonton, Alberta.
11. Maynard, D.; Tablan, V.; Bontcheva, K.; Cunningham, H.; Wilks, Y. *MUlti-Source Entity recognition – an Information Extraction System for Diverse Text Types*. Technical report CS--02--03, Univ. of Sheffield, Dep. of CS, 2003. http://gate.ac.uk/gate/doc/papers.html
12. Kiryakov, A.; Popov, B.; Ognyanoff, D.; Manov, D.; Kirilov, A.; Goranov, M. *Semantic Annotation, Indexing, and Retrieval*. To appear in Elsevier's Journal of Web Semantics, Vol. 1, ISWC2003 special issue (2), 2004. http://www.websemanticsjournal.org/
13. University of California, Santa Barbara. *Alexandria Digital Library Feature Type Thesaurus*. Version of July 3, 2002. http://www.alexandria.ucsb.edu/gazetteer/FeatureTypes/ver070302/index.htm
14. Noy, N. *Representing Classes As Property Values on the Semantic Web*. W3C Working Draft 21 July 2004. http://www.w3.org/TR/2004/WD-swbp-classes-as-values-20040721/
15. Davies, J.; Boncheva, K.; Manov, D. *D5.0.1 Ontology Engineering in SEKT (informal)*
16. Laboratory of Applied Ontologies, Institute of Cognitive Science and Technology, Italian National Research Council. *DOLCE: a Descriptive Ontology for Linguistic and Cognitive Engineering*. http://www.loa-cnr.it/DOLCE.html
17. Pinto, S.; Staab, S.; Tempich, C. *DILIGENT: Towards a fine-grained methodology for Distributed Loosely-controllled and evolvInG Engineering of oNTologies*. In Proc. of ECAI-2004, Valencia, August 2004.

## Appendix A: PROTON-specific Axioms

These axioms are built in the x.y.z release of Sesame. Therefore they would not work (at least not without any additional effort) in case PROTON gets to be interpreted by other tools. Such a customization of Jena seems to be fairly strightforward, taking into account the general rule engine that is built in it.

What is valid for both of the axioms is that if the engine does not support them, then the applications should write somewhat more ellaborate queries in order to achieve the same results. There are no other problematic implications on the ontology, so it is portable to any OWL (Lite) compliant tools.

1 The semantics of the **`protons:transitiveOver`** property

Premises:

```
<ppp, protons:transitiveOver, qqq>
<xxx, ppp, yyyy>
<yyy, qqq, zzz>
```

Consequent:

```
<xxx, ppp, zzz>
```

---

2 Inference of **`protont:involvedIn`** about **`protont:Agent`** instances that 'hold' a **`protont:Role`** in some **`protont:Happenning`**

Premise:

```
<xxx, protont:roleHolder, yyy>
<xxx, protont:roleIn, zzz>
<yyy, rdf:type, protont:Agent>
```

Consequent:

```
<yyy, protont:involvedIn, zzz>
```