Project Number	IST-2006-033789
Project Title	Planets
Title of Deliverable	*Specification of a Planets-wide Ontology of properties for digital preservation needs*. (Part three of a three-part final report from the Digital Object Properties Working Group Report)
Deliverable Number	D23C
Contributing Sub-project and Work-package	PC/3
Deliverable Dissemination Level	External
Deliverable Nature	Report
Contractual Delivery Date	26th April 2010
Actual Delivery Date	26th May 2010
Author(s)	UzK

Contributors

Person	Role	Partner	Contribution
Robert Kummer		UZK	Author
Johanna Puhl		UZK	Author

Keyword list: Ontology, XCL, property, observational, extractable, **p**reservation, rendering, performance, OWL

Contents

1. Purpose of this document

The aim of this document is to set the context of the ontology work that has been undertaken during the course of PLANETS and which has lead to the development of the PLANETS-wide ontology.

This report is part of a three-part final report from the PLANETS Digital Object Properties Working Group. The three companion reports, which can be read in conjunction, are:

- *The concept of significant properties.* (PLANETS deliverable PC3 – D23A);
- *Planets components for the extraction and evaluation of digital object properties* (PLANETS deliverable PC3 – D23B); and
- *Specification of a Planets-wide Ontology of properties for digital preservation needs.* (PLANETS deliverable PC3 – D23C) (this report).

2. Previous work

Within the PLANETS-project a lot of investigations regarding preservation-relevant (file-) properties have been pursued in different sub-projects. To unify these different achievements a model was needed that represents the outcome of this researches and relates them to each other. It has shown that for this purpose an ontology can be an effective tool to not only unify the results of research but also link them to each other to be able to generate new knowledge about a certain domain of interest.[1]

Research on ontologies within PLANETS started with developing the XCL ontology. It has been designed at the University of Cologne to establish a canonical naming scheme and conventions for classification of file-format properties within extractor and comparator tools. These pieces of software and two XML-based languages have been created in the PLANETS sub-project “Preservation Characterisation” in Cologne.

The extractor is supposed to read all available information within a file (of one of the supported formats) and to create a description file that contains this information. The description file is written in the XCDL-language that is easily readable for humans and machines.

In a scenario where a file has been migrated to a different file-format and a preservation-manager needs to find out if all information has been correctly transformed into the new file (of the new format) he can find out by having the information of both files extracted and encoded in the XCDL-format. Then, another tool that has been built in Cologne, the Comparator, can scan both files for properties that differ after being migrated from one file format to the other.

The XCL-Ontology takes care that all property are denominated in a way that they conform to a certain naming convention as well as are relying on the same units and data types. Both the extractor and comparator query the ontology by accessing a web service endpoint and by then retrieving the according information to map file-specific names to the canonical ones.

3. Ontology Sources

[1] “Ontology” is a term frequently used in Semantic Web research. For more information about the concepts used in this document refer to the W3C Semantic Web page (http://www.w3.org/). It provides extensive information and introductory material.

Complementary to the XCL-Ontology, another ontology has been established, the PLANETS-wide ontology. While the former is focused on dealing with file properties, the latter also aims at including additional properties that are significant in the preservation context. It should represent the knowledge about properties dealing with digital preservation in general that turned out to be important in different PLANETS sub-projects. Consequently, it relies on the XCL-Ontology as a foundation but extends the structure beyond its means to encode file properties.

To facilitate distributed development while maintaining a clean structure, the PLANETS-wide ontology has been split into several dependent ontologies that are maintained in different files. The relations between properties that stem from different projects and therefore appear in different files need to be investigated, defined and modelled.

The following information has already been included within the Planets-wide Ontology:

- PP/2: Report on policy and strategy models for libraries, archives and data centres[2]
- Plato: Preservation plan template for significant properties[3]
- TB/3: Methods for testing[4]
- Inspect: Properties from the INSPECT project[5]
- PC3-D9 part 2: Classification scheme for representation information networks[6]
- XCL-Properties: Properties from the PC-Extractor and -Comparator[7]

The file properties that have been listed in each of the sources used were simply integrated into subclasses according to either the subprojects name or – in the case that further classifications were made – the classifications name. For example, the work of TB/3 lists properties according to kinds of file types similarly to the XCL-structure. [did exploit information if it was available]

The semantic structure that was used to build the sub-ontologies was therefore the same as in the XCL-Ontology to be able to create links between these in an uncomplicated way.

Still, not all semantic issues could be solved according to the slightly different meaning of properties from different subprojects with almost the same names.

4. Ontology Purpose

Many properties that stem from files and other preservation topics have been discussed in the scope of the PLANETS project. To provide an overview, a model needed to be established that is both machine- and human-readable. This is guaranteed by using OWL as XML-RDF representation for the Ontology.

---

[2] http://www.planets-project.eu/private/pages/wiki/index.php/PP/2_Draft_Conceptual_Model

[3] http://gforge.planets-project.eu/svn/planningtool/trunk/data/templates/public_fragments/Significant%20properties.mm

[4] ftp://www.planets-project.eu:1924/docs/Deliverables/5Testbed_(TB)/Planets_TB3-D2_MethodsforTesting_final.pdf)

[5] http://www.significantproperties.org.uk/resources.html

[6] http://www.planets-project.eu/private/planets-ftp/WP_PC/XCLOntologyJP2HMrdf.owl

[7] http://planetarium.hki.uni-koeln.de/planets_cms/ontology/XCLOntology.owl

Machine-readable in this context means that OWL can be interpreted through reasoning software and therefore enables machines to draw conclusions from it.

The XCL-Ontology has already been used within the “Preservation Characterization” project and the latest version of the test bed environment. There, it proved to be a useful tool.

Basically, ontologies have been proposed to support communication processes in larger groups. They have been developed to help organisations finding a common language and understanding of important concepts. In comparison to flat glossaries or terminology lists, ontologies can have a complex thesaurus-like structure. Thus, they not only define certain notions but can also encode complex interrelations.

File properties can be most adequately described together with their surrounding preservation ecosystem. Thus, the ontology seems to be a good way to describe them. In certain use cases ontologies can help the manager of a preservation-workflow to decide on which tools to use and learn which sequences of operations have to be executed.

A somewhat more elaborate description on how an ontology can be used has been formulated as a user scenario[8]. It describes certain situations and user actions where the utilization of the PLANETS-wide ontology most likely could be helpful.

Institutions that take care of the long-term preservation of digital objects rely on comprehensive domain knowledge. Here, the ontology will function as a knowledge base with structured information about file properties and those that are also important in a preservation ecosystem. For example, in the context of PLANETS, the ontology will be of use for the Plato planning tool and the Testbed, once it is implemented.

As it provides the user with information about file-properties and properties that are important in a preservation ecosystem it can be very helpful in an institution that takes care about the long-term preservation of digital objects.

[8] This scenario can be found in companion report, *Planets components for the extraction and evaluation of digital object properties* (PLANETS deliverable PC3 – D23B)

5. Standards to be used

The Planets-wide Ontology will be written in OWL/RDF with the expressivity of OWL-Lite. In general, there are three different ways to model an ontology using the following dialects: OWL Full, OWL Lite and OWL DL (= Description Logic). Those dialects relate to each other as follows: A document written in OWL Lite is also a valid OWL DL-document, which is also a valid OWL-Full document. This means that OWL Lite is the most specialised dialect of these three but still appeared to be suitable to express preservation topics. The described order expresses that OWL Full has the loosest rules and allows constructs that cannot be handled by common Semantic Web tools. OWL Full can handle almost any construction as long it is valid RDF and is therefore most endangered of being used in an inconsistent way.

As OWL Lite is the strictest conception of this order, it does not allow many of the concepts that the other OWL dialects do allow[9].

5.1. Ontology integration

Each of the ontologies has been modelled in a way that supports its alignment with the PLANETS-wide ontology. OWL and RDFS do provide several ways to align information that has been modelled in different ontologies. These comprise SWRL[10] rules; OWL and RDFS class axioms as well as additional means to express identity of individuals.

To align classes properly that have been defined in different project ontologies the RDFS property rdfs:subClassOf has been used. Most Semantic Web software will be able to handle this property for correct reasoning. RDFS has been preferred to OWL (owl:equivalentClass) wherever it was possible for reason of compatibility and simplicity:

```
@prefix rdfs: <http://[…]/22-rdf-syntax-ns#>.
@prefix xcl: <http://[…]/XCLOntology#>.
@base <http://[…]/PLANETSOntology#>.

:audioInformation a rdfs:Class;
        rdfs:subClassOf xcl:XCL_Properties;
        rdfs:subClassOf :ExtractableProperties.
```

[9] See also: http://www.w3.org/TR/owl-features/

[10] SWRL = Semantic Web Rule Language: http://www.daml.org/2003/11/swrl/

The built-in OWL property owl:sameAs can be used to link one OWL individual to another individual. However, in the scope of XCL it seemed to be reasonable to avoid this mechanism as properties meanings overlap between different file-formats or use different units or datatypes. So it did not seem appropriate to use a mechanism that implies a perfectly equivalent relationship; therefore we considered it to be helpful to define a special relationship for expressing relationships between properties called xcl:convertTo. This will also prevent Semantic Web reasoners from automatically processing identity relations and adding additional facts. Thus, the responsibility for processing has been delegated to software that can interpret the XCL namespace. The scope of the property xcl:convertTo has been defined rather general to capture the semantics of most conversion processes. Sub-properties could be defined to capture the nature of more specific migrations. For example xcl:castTo could be introduced for converting an enumeration to a string data type.

Since relationships between PLANETS properties also cannot always be defined clearly as identity, we decided to use that same relationship. This relationship expresses that specific properties express the same, but use different units or datatypes in different contexts.[11] Since these can be converted to each other, the relationship has been called xcl:convertTo.

Note: The terminology regarding properties is a little precarious as OWL itself uses a concept called property (OWL:property) to express relationships between individuals while we decided to model file-properties as individuals. So this is why we tried to avoid the notion “property” for relationships that are in fact modelled as OWL:property.

5.2. Anonymous classes

Anonymous classes are unnamed classes that may be needed to relate other classes to each other through concepts like owl:unionOf. OWL Full completely supports anonymous classes while OWL DL only accepts anonymous classes if they are not in a domain-range, equivalent- or disjoint- relationship to other classes. OWL Lite supports anonymous classes in an even more limited way. All classes and sublasses need to be named before one is able to define relationships between them. This is completely acceptable for our purpose, as we define properties to have very clear origins and do not use classes to define their internal relations.

5.3. Non-transitive cardinality-restrictions:

Non-transitive cardinality-restrictions are useful in situations where A relates to B but not the other way around. Here, it is not possible to express cardinalities of any kind. Restriction rules are also valid for modelling OWL DL ontologies.

The following rules are only restrictions for OWL Lite but not for OWL DL:

- Constructs like owl:oneOf, owl:disjointWith, owl:unionOf, owl:complementOf and owl:hasValue are not allowed.
- Cardinality Expressions can only have the values 0 and 1.
- owl:equivalentClass is only usable between named classes.

For further information on the different dialects see[12].

[11] http://planetarium.hki.uni-koeln.de/planets_cms/sites/default/files/PC2D12D13PC4D7-01.pdf

[12] http://www.w3.org/TR/owl-features/

In opposite to the other OWL dialects OWL-Lite is also calculable in finite time. In the scope of PLANETS, the advantage of OWL Lite is that it doesn't allow the conceptual merging of individuals (entities) and classes. This means that in class hierarchies, subclasses are always still abstract concepts and can never be real – concrete – instances of classes – this supports the differentiation between a file-property itself (bitdepth) and the abstract concept of a property (imageproperty). This conceptual separation also allows for a specified kind of reasoning and is therefore well qualified for logical conclusions.

Regarding logical conclusions, the Ontology can be queried by SPARQL services. This is helpful for advanced usage in the PLANETS-software Testbed and PLATO.

6. The Structure

6.1. The basic concepts: Preservation concepts and file properties as classes and individuals

This section will describe the structure of the PLANETS-wide ontology. Concepts that are important in the context of preservation activities have been modelled as so-called classes (the appropriate OWL concept is owl:Class). By contrast, file properties have been constructed as so-called individuals (owl:Individual) that belong to a certain class (for example xcl:XCL_Properties). From this follows, that they are particular materialisations of an abstract concept.

Still a file property is an abstract concept (e.g. imagewidth is an abstract concept, that is an instance of the class image-properties) in the PLANETS-Ontology. As a file property is already constructed as the lowest leaf within the hierarchical order of the Ontology, it cannot contain a concrete value for a certain files. It just states that a file of a certain type containes properties that can have values of a certain type.

Individuals (i.e. file properties in the context of PLANETS) can relate to other individuals like measurement units that do apply to a certain file property. Additional information on the distinction between owl:Class and owl:Individual can be found in the W3C section "Design for Use" in the document "OWL Web Ontology Language Guide".[13]

6.2. Describing file properties with units and datatypes

Other constructions are the aforementioned units or data types, because technical extractable properties like the width of an image (imagewidth) do have a data type like integer and units like pixels. Other properties do only have a data type and no unit. A good example is a file name that has string as its data type but no unit like centimetres (cm) or beats per minute (bpm).

The constructions "datatype" and "unit" are modelled as an annotation property in OWL (owl:annotationProperty). Their instances (e.g.: bpm, cm, inch, em for unit or integer, string, bool for datatype) are also constructed as individuals because they are again particular instances of an abstract concept data types respectively units.

Each Property can be associated with a unit and a datatype, if necessary, and intended in the original file-format. It can be helpful to define mapping-mechanisms between units like centimetre and inch through the OWL object property construction (owl:ObjectProperty).

These are algorithms that have already been defined in the XCL-Ontology like the conversion from em to point (planets:em_to_point) or centimetres to pixel (planets:cm_to_pixel).

[13] http://www.w3.org/TR/2004/REC-owl-guide-20040210/#DesignForUse

6.3. The concept of OWL:ObjectProperty

Object properties will be used to define relations between PLANETS-properties. Although they carry a similar name, they will not be used to define PLANETS-properties themselves. The recommended usage in OWL is to map entities as referred to above.

Object properties will also be used to map properties from different provenances to each other. This allows for managing redundancies. For example:

The property for the color depth of an image is defined within the PLANETS-Subproject TB/3 (tb3:Color_depth). It can also be found within a lot of file-format-standards, like TIFF or JPEG. In this case, the counterpart of this property will be available in the XCL-Ontology, that already contains a lot of extractable Properties. These are already mapped by object properties to the XCL-naming-conventions.

In this case the object property is called xcl:convertTo. This facilitates to use the same object property again to map the TB/3-Property tb3:Color_depth ("Describes the number of bits used to represent the colour of a single pixel; Can not be determined by visual inspection (you cannot simply see it) but only by a tool.") to the XCL-Property xcl:bitDepth which denotes the number of bits per colour. Unfortunately, this example illustrates one of the rare cases that do not require further attention in opposition to a property like an author of a certain source.

There are several properties that need much more attention like the aforementioned property that denotes an author in PP2 (pp2:author). We still have to define relations that make it possible to link this property to a property like "Creator" because both properties can denote the same entity but do not necessarily. Therefore, the defined relationship has to interact with other relationships. "Author" and "creator" should be connected by a relation that indicates their sameness in some contexts (for example xcl:convertTo) but also by one that expresses their inequality in other contexts, which has not yet been defined.

This example should clarify that properties need to be well defined to prevent the ontology from becoming inconsistent. Up to now, the particular ontologies have not undergone in-depth investigation regarding the aforementioned criteria.

7. The Terminology

7.1. Common understanding of "Property" and "Characteristic"

Referring to the paper of Angela Dappert et al.[14], we decide to denominate the entities found in file-formats as "properties", while concrete instances of these (a certain tangible value of a property in a certain file) are called "characteristics". As all the ontologies subsumed in the PLANETS-ontology are abstract concepts of properties (i.e. they don't carry any concrete values for certain files), the term to be used within these ontologies is always "property".

7.2. Observational and Extractable in the Planets Ontology

To find a common understanding of properties worth maintaining, we have identified two categories. By categorizing file- and environmental properties as "extractable" and "observational", the user-experience of a file should be preserved.

[14] Angela Dappert and Adam Farquhar: Significant Properties, Characteristics, or Requirements. Page 3

During the investigation of properties in the PLANETS project, the category "extractable" was brought up by software like the XCL-Extractor. Referring to this strategy of extracting properties a terminology problem emerged, on how to name properties, which are not extractable by software. We decided to call these "Observational". In order to find a clear structure on which to map properties to in the Ontology both terms needed a clearer definition.

7.3. Definition of Extractable Properties

A property falls in the category “extractable, if it is kept within a file and can be extracted by a common peace of software. In contrast, there exist other properties that relate to a file but are not stored within the file. Therefore, they are not easily extractable. The latter fall in the category “observation” which is also referred to as “observable”. However, both denote the same thing.

A property such as "image-resolution" clearly is extractable – its value can be drawn from a file's bit stream itself. Additionally, the resolution of an image is clearly not observable; a human eye can’t measure that an image has the exact resolution of 300 dpi. If any the human observer can only judge whether image 1 has a higher resolution than image 2 or not. On the other hand, one could argument that a human observer will consult software like Photoshop to figure out the resolution of an image. But in this case, it is clear that the software still extracts the resolution – the human observer still has to rely on the software that informs about the value of the resolution.

In some cases it is difficult to define and distinguish these categories. One example for why it is so difficult to define the category "extractable" is the property "operation system". In general, one would assume that information about the file-system on which a file was produced could be easily extracted by software. Files can be produced in a lot of file-formats on different kinds of operating systems. Most file formats do not store that information and the operating system cannot be considered to be “extractable” in the sense of the narrow definition. The following two examples should help to better understand the distinction.

Neither PDF nor TIFF and JPG store information about the operating system context according to their format-specifications. But there surely exist file-formats – in most cases metadata formats – that store that information, these are for example: The NISO-standard for describing images. This standard stores a property called hostComputer, which is described as "Computer and/or operating system in use at the time of image creation.” So we can assume, that in the case of a NISO-MIX-file information about the creating operating system is extractable.

Another example is the property that represents the quality of an image. If we define the category “extractable” wide enough, we could find a piece of software that is able to identify the quality of an image and encode it as a value on a certain scale by measuring the grain size within areas of an image. Still, this property, image-quality, heavily depends on what criteria it is measured with and by which piece of software it is extracted. Usually a COMMON[15] image-rendering programme is not able to decide how high the quality of an image is, nor do file-formats for images store this information within themselves. Therefore, we consider the image quality to be not “extractable” but rather “observable” by a human. And therefore it belongs to the category "observational".

[15] We define common software to be a programme that can be operated by an average computer-user (e.g. no command-line-tools)

7.4. Definition of Observational Properties

"Observational" properties are properties that relate to files but are not stored within the files and are therefore often not "extractable". An example for this category of properties is clearly the above described image quality as well as information about the regulation of legal access. This property could contain any information about a license or regulation connected to the access to a file and is important in a preservation context within an organisation. Still there is no common software that is able to extract this information neither from a file nor from the software environment.

There are cases where it is not so easy to decide whether a property is "observational" or "extractable". As shown above, the quality of an image can be an example for conflicts; another one is page number information in text files.

Obviously, a human observer can comprehend a page number. And it can be extracted by software in certain cases. For example, file formats from the Microsoft Office-family, like DOC, DOCX or RTF store information about page numbers in a certain field for each page. This field usually contains the integer-value that is interpreted as page-number. But a file-format like PDF does not store structural information like heads, subheads, chapters or page-numbers as properties.

Within a PDF document a page-number (=the number on the page itself) is not interpreted as a page-number but as just another text-field with the coordinates in the bottom of the page. The problem of unsynchronised page-numbers in PDF is known. However, there is still no software that would be able to solve that problem. Therefore, page numbers are only "observational" for PDF documents but can also be "extractable" for other formats like MS Office files.

7.5. Approach to Terminological Problems

7.5.1. Observational vs. Extractable

One approach to the terminological problems that have been described is to define the categories “observational" and “extractable" as not being mutually exclusive. The definition should aim at helping the operator of preservation-software to decide whether he should use software to compare properties or whether he should compare properties manually. He can rely on software if the property is “extractable” and needs to allocate human resources if a property is “observational”. To put it in a nutshell, we constitute that a property is "extractable" if it is kept within a file and is extractable by a COMMON piece of software.

Within the Planets Ontology the properties in those two categories will therefore have no disjunctive relation but rather the relation of an intersection.

7.5.2. Rendering properties

The advantage of the Planets-ontology is that it can explicitly define, which software is able to extract properties from files.

One could model the perceived quality of an image as a property that belongs to the class for properties relating to graphics and leave it at that. Additionally, since OWL supports multiple inheritance, this property could also belong to a further class for properties that can only be extracted after rendering a file. But by making use of object properties more complex coherences can be modelled that better reflect how certain properties work. The property that represents the image quality could be further refined within the ontology to express that it is extractable with a certain software or algorithm and is related to the current graphic board of the local computer.

By advancing this approach, the ontology could become a knowledge base that helps software to assess whether it is possible to extract a property and under which circumstances. The technique described is planned for being implemented into the XCL-Ontology and will be integrated in the PLANETS-ontology as well. This will allow file creators to construct detailed statements about properties and their interactions with software. The class that holds properties that depend on a file to be rendered will be modelled parallel to “observational” and “extractable” properties. Thus, they inherit from the same class that holds file properties (planets:FileProperty). Time will tell if this construction is sufficient for future use cases or if further discussion needs to take place for better clarification. But ontologies tend to be flexible in adopting changes, which can easily be made by moving subclasses to different levels within the class hierarchy.

7.5.3. Performance Properties

In the area of software engineering performance is defined as the ability of a software environment to conduct certain tasks within a certain amount of time and according to other criteria like synchronicity, non-intermittent duty.

Apart from the information stored within a file, there is additional information needed to calculate statements about the "performance" of files/software/operation systems like the processor-type, the amount of other programs running on the machine at the same time etc.

Regarding all these criteria, that are needed to extract the performance of a certain piece of software we can consider that software, that measures performance is a) not common in use and b) the information about performance properties is not contained in the objects themselves. "Performance properties" are rather a subclass of "Observational" properties in the context of the Ontology but will be treated the same way as “Rendering Properties”. At the monent it is not foreseeable to which extent these properties will be dicussed within the last months of PLANETS.

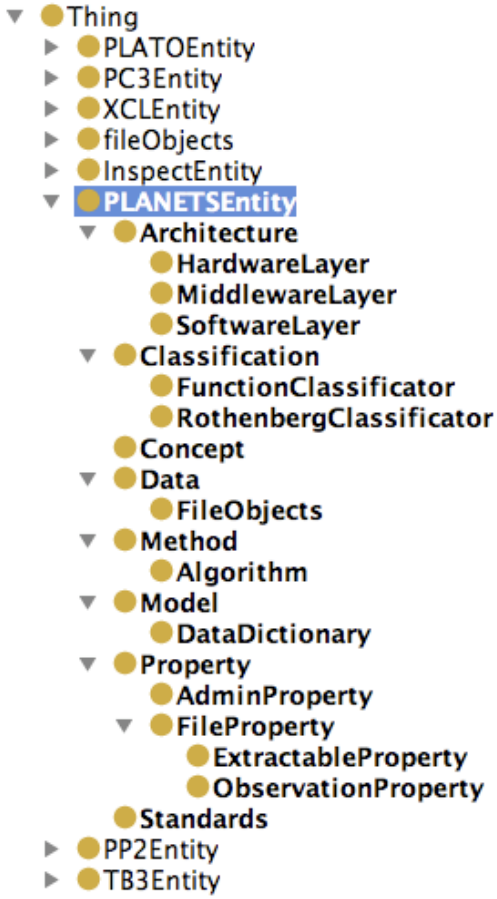
8. Technical documentation

8.1. The Planets-Ontology as a whole

The PLANETS-ontology is modelled in OWL-Lite but does contain only a very few individuals (PLANETS file properties) itself, because these are integrated from different dependent ontologies.

The ontology itself contains the basic class structure that incorporates different kinds of properties for different purposes as described in the several Planets sub-projects and in the terminology chapter in this paper.

A screenshot of the class structure of the Planets-Ontology:



The efforts of each sub-project that have been mentioned in chapter 3 have been modelled as an individual OWL-file. Each file shares the same design principles where properties have been modelled as members of subclasses. The sub classes in these project-specific ontologies have been constructed according to their description in the subprojects. The specific ontologies are aligned with the PLANETS-wide ontology by mapping project specific classes to their global correspondents. As this is not possible with all classes, some properties (individuals) have to be manually asserted to new classes, like the to categories “extractable properties” and “observable properties”.

Determined by its historical development one of the PLANETS-wide concepts, the units and data types will be kept in the XCL-Ontology, to maintain its independence as a single file. But still these concepts can be used within the whole PLANETS-ontology and the other project-specific ontologies without having to declare them again. This is done through the mechanism of incorporation of other ontologies: The ontology that has been generated from information in TB/3, for example, includes the XCL-Ontology. By exploiting this mechanism, units and data types that have been defined in XCL can also be used in TB3. This case might explain why the provenance of properties influences the structure of it.

8.2. PLATO issues

The provenance of properties did raise some questions while aligning the PLATO-ontology to the PLANETS-ontology. PLATO offers certain properties to a user to let him assign a priority value to it for the later execution of a preservation plan. The provenance of these properties has never been completely clarified. Some of them certainly stem from XCL (because Plato uses the XCL-Comperator); others remind of planning-properties, which have been discussed within PP/2 – again, others seem to be new. Therefore, we decided to include only the properties exclusively found in PLATO. These comprise for example properties that have been defined to describe console video games. Others have been omitted to avoid redundancies.

The Ontology will be stored at http://gforge.planets-project.eu/svn/xcltools/trunk/PLANETS_Ontology/ until the end of the project[16]. It will be edited with protegé 4.0 Beta. Some documentation has been put on-line as interlinked html-pages at "http://planetarium.hki.uni-koeln.de/planets_cms/ontology/owlDoc/index.html".

9. Outlook

Within the ontologies, all properties will be arranged into the classes "extractable" and "observable" and – depending on the ongoing discussions – into "rendering". Then, they will be checked for redundancies with respect to their semantics.

Three environments have been identified where the ontology will be useful:

- XCL, PLATO and Testbed have already incorporated the ontology into their software to support certain actions. For example, the extractor tool of XCL consults the ontology to derive canonical names for file properties.
- Therefore, these projects will act as test cases for further development according to the user scenario document. E.g., it could serve as one component of an expert system that helps preservation managers to obtain information on risks for file-formats and tools.
- Finally, it could serve as a basic knowledge basis for the development of new file-formats and tools, since it is supposed to hold comprehensive knowledge about files and their environments. This could deter the developers of file-formats from introducing strange new units and properties and could also prevent them to define mistakable properties like "author" and "creator" or "bitDepth" and "colourDepth".

Nevertheless, it is essential to emphasise that the boon and bane of ontologies is, that their contents are always expansible and they are never complete. The PLANETS-ontology therefore only reflects the current state of discussions in its class-structure and – considering the hundreds of file formats, tools and other criteria for preservation-decisions – its contents will probably never be plenary.

[16] Log in with username: anonymous and password: empty.