Implementing Metadata that Guides Digital Preservation Services

Angela Dappert & Adam Farquhar

British Library, Boston Spa, Wetherby, West Yorkshire, LS23 7BQ, UK
Angela.Dappert@bl.uk, Adam.Farquhar@bl.uk

Abstract

Effective digital preservation depends on a set of preservation services that work together to ensure that digital objects can be preserved for the long-term. These services need digital preservation metadata, in particular, descriptions of the properties that digital objects may have and descriptions of the requirements that guide digital preservation services. This paper analyzes how these services interact and use this metadata and develops a data dictionary to support them.

1 Introduction

Effective digital preservation requires a set of preservation services that work together to ensure that digital objects can be kept alive for the long-term. In order to work together, these services need digital preservation metadata such as descriptions of the properties that digital objects may have and descriptions of the requirements that guide digital preservation services. This paper analyzes how these services interact and use this metadata. From this it develops a data dictionary to support them.

1.1 Related Work

Digital preservation metadata is the information that is essential to ensure long-term accessibility of digital resources. Analyses of the goals of long-term digital preservation have led to a solid understanding of the types of metadata that are needed. Good overviews are provided in Caplan [3] and Lavoie [13]. In 2002, OAIS [4] provided a framework to unify the concepts and terminology in the community. Its information model [19] defines categories for preservation metadata. In 2005 the PREMIS data dictionary consolidated several earlier efforts [e.g. 5, 14, 17, 18] to produce conceptual models and concrete metadata dictionaries for implementers of digital preservation services. Now in its second version [20], it has been widely accepted and plays a key role in creating coherence in the digital preservation metadata community. PREMIS provides a foundation to support interoperability across systems and organizations. Many of the entries in today's data dictionaries are, however, still vague. They await increased practical experience to establish the proper level of granularity. They also tend to be focused on statically recording characteristics and events rather than on dynamically supporting preservation processes.

1.2 Contributions

This paper draws on the practical experience gained in Planets [10], a four-year project co-funded by the European Union to address core digital preservation challenges. It analyzes how preservation services interact and use preservation metadata. From this, it derives information needed to capture key preservation metadata elements, such as property, characteristic, and requirement. Finally, it develops a data dictionary to support the analysis. The approach handles conflicting values from multiple sources. It also supports dynamic preservation processes, in addition to static recording of characteristics and events. It is based on a conceptual model of digital preservation that is theoretically and empirically founded [7, 8]. The model has consequences for implementations of preservation metadata dictionaries, property registries, and preservation services.

2 Properties, Values, Characteristics and Requirements

In order to write with a reasonable level of precision, we need to introduce a basic vocabulary [6]:

- Entity – Anything whatsoever.
- Class – A class is a set of entities. Each of the entities in a class is said to be an instance of the class.
- Individual – Entities that are not classes are referred to as individuals.
- Property – A property is an individual that names a relationship.
- Characteristic – A property / value pair associated with an entity. The value is an entity.
- Facet – A facet is a property / value pair associated with a characteristic. The value is an entity.
- Constraint – A Boolean condition involving expressions on entities.
- Requirement – A constraint in a specific context.

Unless otherwise specified, a characteristic is directly associated with entities. Furthermore, we say that a property applies to classes if it can be meaningfully associated with some instances of these classes.

We can use this language in the domain of digital objects and preservation. For example, file is a class; f1.txt is an instance of the class file; fileSize is a property; the property fileSize applies to file; the file f1.txt has the characteristic fileSize = 131342.

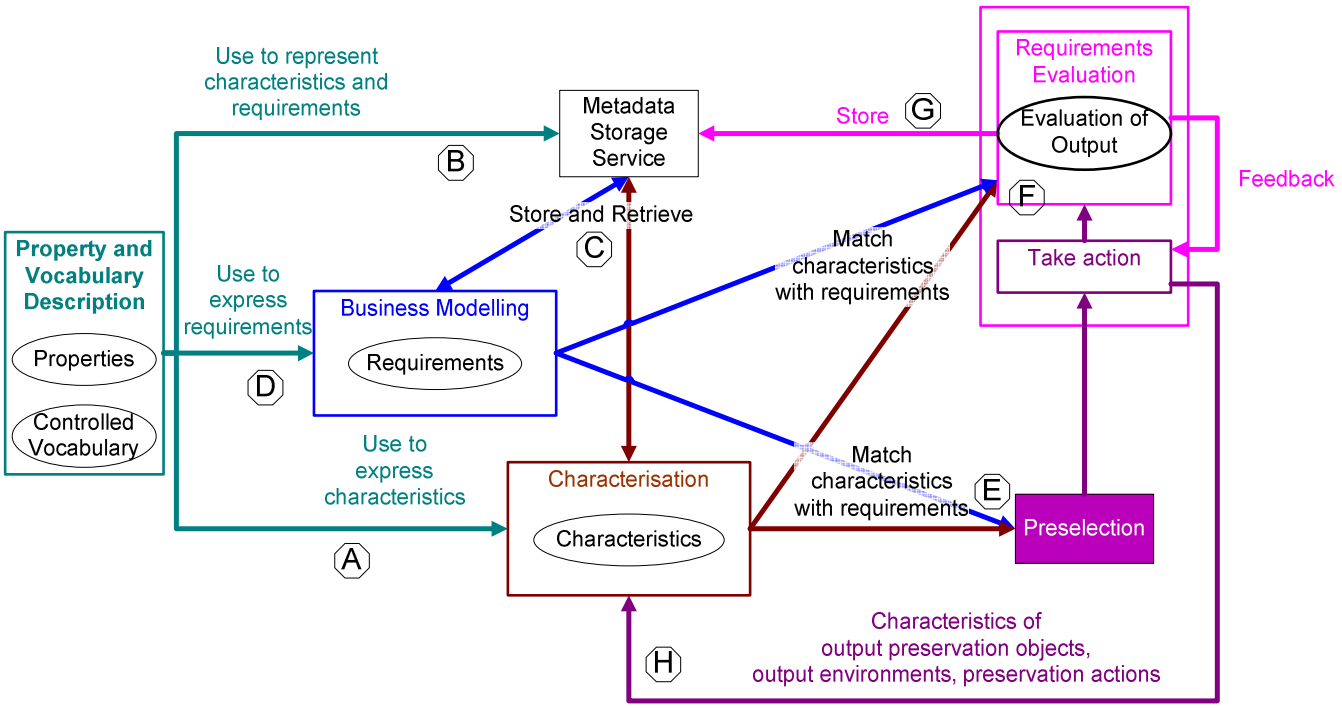

Figure 1 shows how properties, characteristics and requirements interact

The constraint language can be used to express richer relationships. For example, suppose a is a bitPreservationAction, fIn is the initial file, and fOut is the result of applying action a to fIn, then the constraint fileSize(fIn) = fileSize(fOut) should hold.

Important additional information about a characteristic, such as how a value is encoded, the unit of measure, or the algorithm or tool used to compute it can be specified using facets.

The core classes in the digital preservation domain are preservationObject, preservationAction and environment.

The preservationObject concept corresponds to those objects in need of preservation. In our conceptual model [7, 8, 9] it has the subclasses bitstreams (including bytestreams and files), representations of logical objects consisting of representation bitstreams that are needed to create a single rendition of a logical object, and logical objects such as intellectualEntities and components. An intellectualEntity is a distinct intellectual or artistic creation, a set of content that is considered a single intellectual unit for purposes of management and description. Finer grained components of an IntellectualEntity are needed to characterise its parts.

The preservationAction concept corresponds to actions taken by custodians of digital content to mitigate the risks that they identify.

The environment concept corresponds to hardware and software environments, the community, budgetary factors, the legal system, and other internal and external factors. An environment or sub-environment can be associated with a preservationObject or preservationAction.

3 USES

Figure 1 illustrates the roles that properties, values, characteristics and requirements (represented by ovals) play in preservation services (represented by boxes). By analyzing the specific roles that they play in these services, we can derive additional requirements for our data dictionary. This will be discussed in the following sections.

3.1 Uses of Properties and Controlled Vocabulary

Properties and controlled vocabulary can be captured in registries so that they can be referred to in other services. Alternatively they can be defined locally for local use in a system. File format registries, such as PRONOM [15] or UDFR [22], can associate file formats with their applicable properties. Characteristics extraction languages, such as XCEL [21], additionally describe how values for these properties can be extracted from files in a given format. Preservation metadata dictionaries, such as PREMIS [20], define common preservation metadata elements to describe properties of preservation objects or environments. Controlled vocabulary registries, such as the planned Authorities and Vocabularies service of the Library of Congress, capture these properties' permissible values (Figure 2). We can use this information to

- link a format to characterization services that can determine values for its applicable properties - for example, a

service to determine the fonts used in a *.doc*[1] file (Figures 1A, 3).

- create a testbed service that measures the degree to which applicable properties are preserved by preservation services - for example, measure the degree to which a service preserves imageWidth by evaluating it on many objects. In addition to the service characteristics (e.g. preservesImageWidth = "no") it could capture the degree to which or under what condition this characteristic holds (Figures 1A, 3).
- enable metadata storage services to refer to properties unambiguously and ensure interoperability and exchange across institutions and systems (Figure 1B).
- identify properties that are shared across file formats and can therefore be preserved by a migration between them (Figure 2).

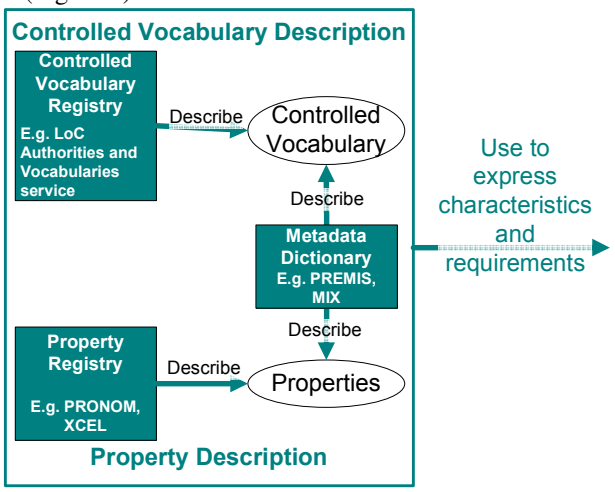

Figure 2: Properties

3.2 Characterization

Characterization services determine the characteristics of preservation objects. Characteristics are property/value pairs. They are used to describe preservation objects, environments, and preservation actions. In particular,

- characteristics can be extracted automatically by characterization tools [11, 16, 21] or assigned manually (Figure 3).
- characteristics of preservation services can be determined experimentally in preservation testbeds [e.g. 1] (Figure 3).
- characteristics may be stored in metadata storage services or produced on demand (Figure 1C).

[1] We refer to file formats via common file extensions as a shorthand for improved readability. A precise statement requires a unique identifier corresponding to an exact version of the format.

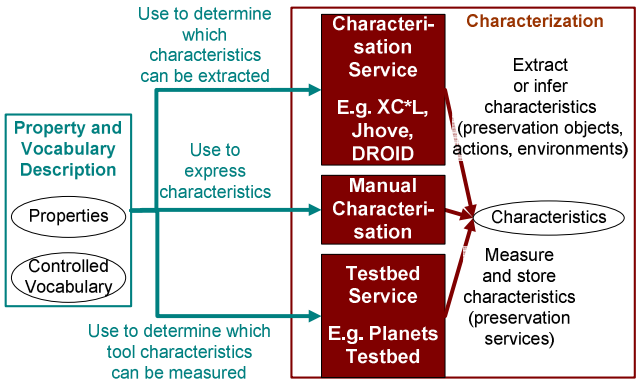

Figure 3: Characterisation

3.3 Business Modelling

Business modelling results in the formulation of requirements from properties and controlled vocabulary (Figure 1D). Requirements reflect the stakeholders' values, goals and constraints with regard to objects and guide preservation services.

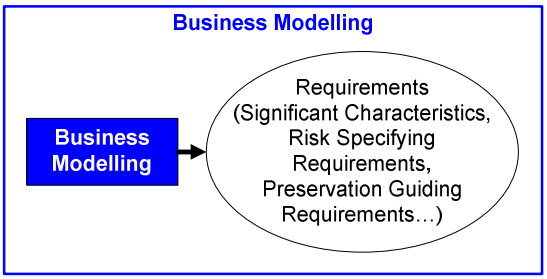

Figure 4: Requirements

They may be captured in preservation guiding documents, such as policy, strategy or business documents. They may also be part of the preservation metadata captured in metadata storage services that documents the constraints that have been or should be applied to specific preservation objects (See Figure 1C). The PREMIS data dictionary for preservation metadata [20] accommodates recording "significant properties" which are a form of preservation guiding requirement. Requirements may also be captured in reusable, customizable user profiles which describe the requirements of a default designated community. (Figure 4)

3.4 Uses of Characteristics and Requirements

Optional pre-selection services (Figure 1E) may provide an optimization step which rules out implausible preservation actions. They analyze requirements to eliminate actions which can from the outset be determined to be violated by characteristics in a given context. Knowledge about the characteristics of preservation services, which has been obtained in testbed services, is particularly helpful in this step.

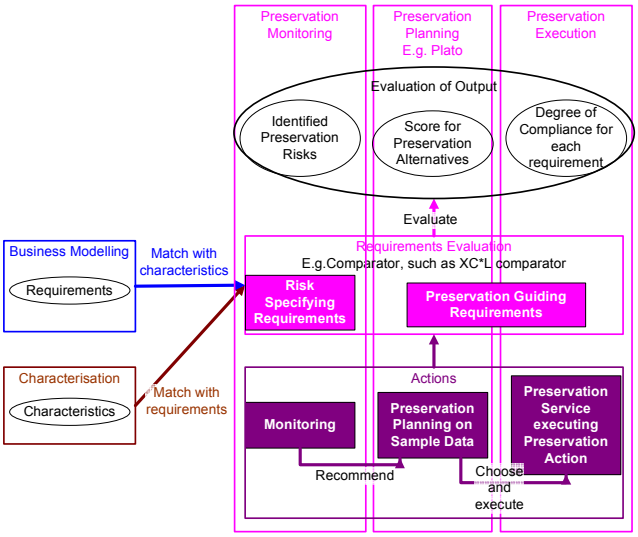

Figure 5: Uses of Characteristics and Requirements

Primarily, requirements guide actions, such as preservation monitoring, preservation planning and preservation execution services (See Figure 5). Preservation monitoring services determine whether risk specifying requirements are violated and, therefore, preservation risks exist. A preservation monitoring process should trigger the preservation planning process once this happens. Using a sample data set, preservation planning services (e.g. Plato, [2]) determine the best choice of preservation service to mitigate this preservation risk, with respect to preservation guiding requirements. The preservation execution service itself uses them to evaluate and validate each preservation action's output.

Once an action, such as preservation monitoring, preservation planning or preservation execution, has been chosen and executed it is validated in a requirements evaluation step. Requirement evaluators [e.g. the XCDL comparator, 21] determine the degree to which characteristics of the preservation objects, preservation actions and environments before, during and after actions comply with requirements. The output is either an assessment of the presence and severity of a preservation risk, or a measure of the degree of compliance of an action with the set of requirements (Figure 1F and 5).

Requirements can also serve as explicit provenance information. A metadata storage service may document the provenance of a repository's objects. For each object, it may record the preservation actions that created it and the set of requirements that applied at the time. It can also store the object's degree of compliance with respect to each requirement in the requirements set, especially its significant characteristics. Sometimes characteristics that are not referenced by any requirement are, however, lost during a preservation action; it is not, in general, possible to record their loss as they can not be listed exhaustively (Figure 1G).

Actions can create new preservation objects and environments. Their characteristics may differ from those of the input preservation objects and environments (Figure 1H). Some requirements may articulate constraints on the relationship between preservation action input and output.

4 SOME OBSERVATIONS

4.1 Observations for Properties

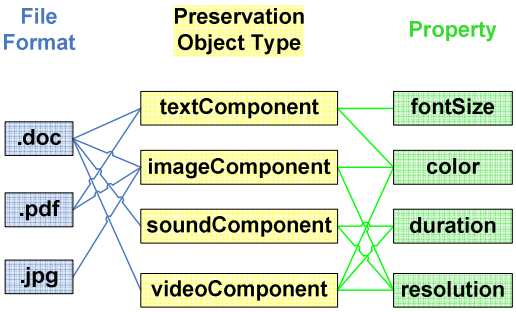

Figure 6: Applicable properties are mapped to formats via the preservation object type

Observation 1:

Many properties are applicable to only a subset of objects. For example, the property fontSize is applicable to formats which may contain text; it would not be applicable to an audio format.[1] In order to achieve a normalized representation, we link properties to the type of class to which it applies (see appliesTo in the data dictionary), rather than directly to file formats. Examples include bytestream, representation, intellectualEntity (e.g. eBook, soundRecording), component (e.g. textComponent, tableOfContents), preservationAction or environment (e.g. legalEnvironment, operatingSystem). This approach makes it easy to express that the fontSize property applies to textComponent objects. Figure 6 illustrates how it is straightforward to map properties to subclasses of component and file formats in turn.

Observation 2:

Properties sometimes refer to a combination of preservation objects, environments, or actions. Consider the relative size of two images, the absolute distance of a line from the text, and the metrics describing column layout. These all refer to several objects. The language that we use to define properties must be expressive enough to capture this.

Observation 3:

Properties are related to each other and their relationships have to be modelled explicitly. For example duration can be calculated from dateTimeRange. Furthermore, many file formats have similar, but not identical properties. Therefore, the language that we use to define properties must be able to capture the relationships between them and specify how to compare or convert them. Figure 7 illustrates this.

[1] The association of properties with digital object types of files is discussed in the Planets testbed [12]. We are refining this to the type of a component of the digital object, since a logical object might well contain, for example, text, sound, and image components together.

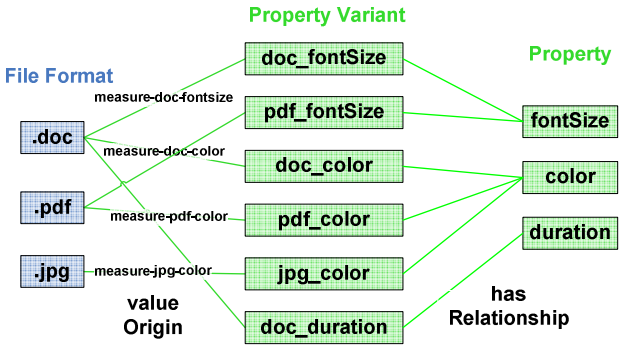

Figure 7: Properties: value origins and relationships between properties

Observation 4:
In many cases, it is useful to define one property in terms of others. For example, the aspectRatio of an image might be defined as imageWidth / imageHeight. As a result, it is essential to record how such properties are defined and derived in order to ensure consistency.
Observation 5:
For each property, it is essential to specify the tool or algorithm that can be used to determine a value and the types of sources from which they can be obtained. We refer to this as the value origin. Values originate when they are

- Assigned manually (stored or on demand). When values are assigned manually they often need to comply with conventions, such as cataloguing rules, standards, controlled vocabularies, etc. This should be specified as part of the value origin.
- Assigned automatically as a side-effect of a service (stored). Regular internal operations, such as ingest of digital objects, purchase of hardware and software, decommissioning of equipment, hiring, training and laying-off of staff, getting and spending money, or executing preservation actions, all change characteristics of preservation objects or their environments. Equally, external operations, such as introducing a new file format or a new preservation service, change characteristics. These value changes need to be captured if they serve as a basis for making preservation decisions.
  E.g. the contentType of objects in an eJournal ingest system is always set to "eJournal" upon ingest.
  E.g. the budget of an institution may be set during the execution of a preservation action: preservationBudgetSize := preservationBudgetSize – preservationActionCost.
- Extracted (stored or on demand). The original source of derived values may be a bitstream or the set of representation bitstreams of a representation of a logical object. Values are extracted using a tool which implements an algorithm. The value origin should specify the algorithms and tools used.
  Examples: bytestreamSize may be extracted from the bytestream object. colorFidelity can be measured by averageColor or by histogramShape. wordCount can count hyphenated words as one or as multiple words. MIME type can be extracted using the JHOVE format characterization tool.
- Inferred (stored or on demand). Values may be inherited in the preservation object hierarchy, derived through a function from values of other properties, or logically inferred.
  The value origin should specify the algorithm that can be used to infer it. E.g. the aspectRatio of an image may be imageWidth / imageHeight.

4.2 Observations for Characteristics

Observation 6:
Values for characteristics may be stored or derived on demand. On demand derivation can take place through characterization services or through retrieval from registries or inventories[1]. Whether they are stored or derived needs to be recorded since different preservation services will be chosen based on this property.
Observation 7:
There may be multiple values for a property of an object, since there may be several representations (sources) which form the basis of measurement for the value and several different measurement techniques (technique) and tools (creation agent). Characteristics and requirements need to specify which value origin is meant.

4.3 Observations for Requirements

Observation 8:
In many cases, a stakeholder may express requirements dependent on additional conditions, e.g. If environmentType = "preservation" then image resolution must be preserved. As a result, the language that we use to define requirements must be expressive enough to include conditionals.
Requirements can be expressed as constraints, such as through OCL [23] or other informal or formal languages.
Observation 9:
Not all requirements are equally important and not all have to be precisely satisfied. To accommodate this, it is useful for a stakeholder to add an importance factor, as a measure of relative importance, and potentially a tolerance factor, as a measure of the tolerable degree of deviation from the specified value, with each requirement. For example, preserving the number of lines on a page might be less important than preserving the number of pages. During requirements evaluation of a preservation action the importance and tolerance factors can be combined into a weighted measure.

5 CONCEPTUAL DETAILS

In this section, we build on the preceding analysis to specify the data model more completely. For each concept, we describe its key attributes and basic information such as its

[1] Such as software licenses, hardware inventories, standards and XML schemata in use, staff skills, etc.

data type and whether it is mandatory or repeatable. We also introduce supplementary concepts such as ValueOrigin and Unit that are needed to represent properties.
This data dictionary is informed by analysis undertaken in the Planets project. It will only be partially implemented during the project, but it serves as a basis for further development and implementation.

5.1 Property

Definition: An abstract attribute, trait or peculiarity suitable for describing a preservation object, action or environment.

- propertyIdentifier (1...1): a unique identifier of the Property (data constraint: Property ID).
- propertyName (0...n): a meaningful human readable name (data constraint: string). It is repeatable in order to allow for synonyms. Different Properties may have the same names, but must have unique identifiers.
- propertyDescription (0...n): a meaningful human readable description of the Property (data constraint: Description)
- appliesTo (1...1): a list of Classes. This property can be meaningfully associated with Instances of these Classes (data constraint: vector of PreservationObject, Environment or PreservationAction subclasses). The vocabulary of subclasses is extensible and includes many subclasses not shown in this paper. See Dappert et al [7] for a sample vocabulary.
- hasRange (0...n): the range of the property. Mathematically, the range is the set of possible values that the property can take on.
  - hasUnit (0...1): (data constraint UnitID).
  - hasDataConstraint (1...1): The range is specified via a constraint, which may be a class, a URI for a defined vocabulary, or a constraint expression. Data constraints are combined with the unit definition, as different units may have different data constraints. (E.g. K: ≥0, °C: ≥ -273.15, °F: ≥ -459.67).
  - isDefault (0...1): indicates whether this is the default range for this Property (data constraint: Boolean)
  - hasDefaultValue (0...1): a default Value for this Property.
- hasValueOrigin (0...n): How the Values for the Property may be obtained or updated (if it is stored).
  - hasValueOriginID (1...1): (data constraint ValueOriginID).
  - isDefault (0...1): indicates whether this ValueOrigin is the default for this Property (data constraint: Boolean)
- hasRelationship (0...n): specify a relationship to another Property.
  - hasRelatedProperty (1...1): (data constraint: Property ID)
  - hasRelationshipType (1...1): a type specification of the relationship to another Property (data constraint: taken from an extensible set; common types include generalizationOf, specializationOf, siblingOf, inverseOf, disjointOf, smallerThan).
- hasEvent (0...n): unique identifiers to each of the Property's Event objects, such as versioning, virus checking, ingest. (data constraint: Event ID).

Value Origin

The ValueOrigin concept provides a way to specify where a specific Value comes from or how it can be obtained. There can be multiple ways of obtaining the Value of a Property that do not produce conflicting results. For example, they might be measured from different sources, measured by different techniques, using different tools, or obtained through different agents.

- valueOriginIdentifier (1...1): a unique identifier of the ValueOrigin (data constraint: none).
- valueOriginName (0...n): a meaningful human readable name (data constraint: string).
- valueOriginDescription (0...n): a meaningful human readable description (data constraint: Description).
- hasSource (0...n): a type specification of the sources from which the Value can be measured or derived (data constraint: none). Sources might be registries or inventories, Values of other Properties from which the Value can be derived, or Representations of the IntellectualEntities from which the Value can be derived. There may be a chain of ValueOrigins where one ValueOrigin is the source for another.
- hasTargetUnit (0...n): a specification of the Unit of the Value to be created by this ValueOrigin. (data constraint: Unit ID)
- hasTechnique (0...n): Rule, algorithm or logic used for obtaining the Value (e.g. assigned according to Anglo-American Cataloguing Rules, extracted from .tiff file metadata) (data constraint: none). Techniques can be manual or automated.
- hasAgent (0...n): For automatically derived Values: software tool and version; for manually assigned Values: person role (data constraint: Agent ID).
- hasTrigger (0...n): a trigger for Value assignment: e.g. ingest, PreservationService, etc. (data constraint: none)

Unit

Every Property can have several Units. This is particularly important for preservation characterization. bitDepth, for example, is described as one non-negative number in .png and as three non-negative numbers (one for every colour channel) in .tiff. It is important to be able to specify which Unit is chosen and how values in this Unit can be compared to others.

- unitIdentifier (1...1): a unique identifier of the Unit (data constraint: none).
- unitName (0...n): (data constraint: string) allows for synonyms; e.g. inches, Zoll.
- unitDescription (0...n): a meaningful human readable description of the Unit (data constraint: Description).
- hasDataConstraint (1...1): permissible Values; a type definition for the Value; possibly a URI for defined vocabulary (data constraint: taken from an extensible set of data constraints).
- hasConversion (0...n): How Values may be converted from another Unit to this Unit. This is important for preservation characterization and comparison.
  - hasSource (1...1): Identifier of the source Unit (data constraint: UnitID)

- hasTechnique (1...n): Rule, algorithm or logic used for mapping or converting the Value (e.g. FFT) (data constraint: none) There may be multiple ways of deriving the Value.
- hasAgent (0...n): conversion software tool and version; (data constraint: Agent ID) There may be multiple possible agents.

5.2 Characteristic

Definition: A Characteristic of an Entity is the concrete Value which this Entity has for an abstract Property in a defined context.

- characteristicIdentifier (1...1): a unique identifier of the Characteristic. Having a unique identifier for a Characteristic supports different Values for the same Property at different times. (data constraint: CharacteristicID)
- associatedWith (1...1): vector of unique identifiers of PreservationObject, Environment or PreservationAction Instances with which the Characteristic is associated. It can be meaningfully associated with Instances of the Classes defined in the appliesTo element of the corresponding Property concept (data constraint: vector of PreservationObject, Environment or PreservationAction IDs).
- hasProperty (1...1): a specification of the Property to which this Characteristic refers.
  - propertyIdentifier (1...1): It specifies for which Property the Characteristic's Value holds (data constraint: Property ID)
  - annotation (0...1): chosen from the allowable values specified in the corresponding Property definition.
    - hasUnit (0...1)
    - hasValueOrigin (0...1)
    - hasSource (0...1)
    - hasTechnique (0...1)
    - hasAgent (0...1)
- isOnDemand (0...1): a specification of whether the Value is stored locally or should be derived on demand (data constraint: one of local, onDemand). Registry look-up is an on-demand access.
- hasValue (0...1): Value of the Characteristic, if it is stored locally (data constraint: none).
- hasCreationEvent (0...1): a unique identifier of the Event which created the Value if it is stored locally (data constraint: EventID) including the date the Value was set. In addition, information to capture versioning information such as a date range of applicability of the Value, previous Values for the same Property and objects, etc. are desirable

5.3 Requirements

Definition: A constraint which limits the space of allowable preservation activities.

- requirementIdentifier (1...1): a unique identifier of the Requirement (data constraint: RequirementID)
- requirementName (0...n): a meaningful human readable name (data constraint: string)
- requirementDescription (0...n): a meaningful human readable description (data constraint: Description).
- hasRequirementsSet (0...n): a unique identifier of the RequirementsSet to which the Requirement belongs (data constraint: PreservationGuidingRequirementsSetID)
- hasStakeholder (0...n): (data constraint: AgentID)
- requirementSource (0...n)
- requirementApplicability (0...1): Time range during which the Requirement is applicable. If it is not specified explicitly, then it defaults to the Value of the applicability element of the PreservationGuidingRequirementsSet in which the Requirement is captured.
  - startDate (0...1): The date the Requirement is projected to become valid (data constraint: date)
  - endDate (0...1): The date the Requirement is projected to cease, if it is not subsequently extended (data constraint: date)
- requirementSpecification (1...1):
  - context (0...n): Specifies the objects for which the constraint holds
  - pre (0...1): Specifies a pre-condition for applying the Requirement
  - post (0...1): Specifies a post-condition for applying the Requirement
- requirementImportanceFactor: Measure of the relative significance of the Requirement for the stakeholder (data constraint: none)
- hasEvent (0...n): unique identifiers to each of the Requirement's Event objects (data constraint: Event ID)

The requirementSpecification can be expressed informally or implemented using a constraint language such as OCL [OCL 2003]. In the latter case, each pre- and post-condition is an expression that can be evaluated against the Characteristic Values specified in the Requirement's context. In some implementations, these will evaluate to simple Boolean values (true or false). Other implementations will allow for a tolerance. In this case, the requirementImportanceFactor and tolerance can be used to compute a weighted measure of compliance with the Requirement.

CONCLUSION

This article has presented a data dictionary for key digital preservation metadata concepts. The underlying conceptual model supports dynamic preservation processes, rather than the static recording of characteristics and events. The data dictionary has been motivated by observations about its intended uses and the interactions between preservation services. The model has consequences for implementations of preservation metadata dictionaries, property registries, and preservation services.

This work has been conducted within the larger context of defining a conceptual model and specific vocabulary for supporting preservation services within the PLANETS project and is theoretically and empirically founded [7, 8].

ACKNOWLEDGEMENTS

Work presented in this paper was carried out as part of the Planets project (IST-033789, http://www.planets-project.eu/) under the Information Society Technologies (IST) Programme of the European Sixth Framework Programme. The authors are solely responsible for the content of this paper.

REFERENCES

[1] Aitken, B (2008). The Planets Testbed: Science for Digital Preservation. The Code4Lib Journal, ISSN 1940-5758, Issue 3, June 2008

[2] Becker, C. et alii (2008). Plato: A Service Oriented Decision Support System for Preservation Planning. JCDL'08, Pittsburgh, Pennsylvania, USA, June 2008

[3] Caplan, P. (2006). DCC Digital Curation Manual, Instalment on "Preservation Metadata", Version 1.0. July 2006.http://www.dcc.ac.uk/resource/curation-manual/chapters/preservation-metadata/preservation-metadata.pdf

[4] CCSDS (2002). Reference Model for an Open Archival Information System (OAIS). CCSDS 650.0-B-1, Blue Book (the full ISO standard). January 2002 http://public.ccsds.org/publications/archive/650x0b1.pdf

[5] CEDARS Project (2002). http://www.leeds.ac.uk/cedars/

[6] Chaudhri, V.,. Farquhar, A., Fikes, R., Karp, P., Rice, J. (1998). OKBC: A programmatic foundation for knowledge base interoperability. In Proceedings of the 1998 National Conference on Artificial Intelligence.

[7] Dappert, A., Ballaux, B., Mayr, M., van Bussel, S. (2008). Report on policy and strategy models for libraries, archives and data centres. PLANETS report PP2-D2. http://www.planets-project.eu/docs/reports/Planets_PP2_D2_ReportOnPolicyAndStrategyModelsM24_Ext.pdf

[8] Dappert, A., Farquhar, A. (2008). Modelling Organisational Goals to Guide Preservation. iPRES 2008: The Fifth International Conference on Preservation of Digital Objects http://www.bl.uk/ipres2008/ipres2008-proceedings.pdf

[9] Dappert, A., Farquhar, A. (2009). Significance is in the Eye of the Stakeholder. ECDL 2009 http://www.planets-project.eu/docs/papers/Dappert_Significant_Characteristics_ECDL2009.pdf

[10] Farquhar, A., and Hockx-Yu, H. Planets: Integrated services for digital preservation. Int. Journal of Digital Curation 2, 2 (November 2007), 88–99

[11] JSTOR and the Harvard University Library. JHOVE - JSTOR/Harvard Object Validation Environment http://hul.harvard.edu/jhove/

[12] Helwig, P. (2007). Test Methods for Testbed. PLANETS report TB/3-D2 http://www.planets-project.eu/publications/?search[0]=9

[13] Lavoie, B., Gartner, R. (2005). "Preservation Metadata". DPC Technology Watch Report No. 05-01: September 2005 http://www.dpconline.org/docs/reports/dpctw05-01.pdf

[14] Lupovici, C., Masanès, J. (2000). Metadata for Long Term Preservation. Bibliothèque Nationale de France. Den Haag : Koninklijke Bibliotheek, 2000. - (NEDLIB Report series ; 2).- With summary ISBN 90-62-59-1469 http://nedlib.kb.nl/results/NEDLIBmetadata.pdf

[15] The National Archives: PRONOM http://www.nationalarchives.gov.uk/pronom/

[16] The National Archives: DROID http://droid.sourceforge.net/wiki/index.php/Introduction

[17] The National Library of Australia (1999). Preservation Metadata for Digital Collections. October 1999. http://www.nla.gov.au/preserve/pmeta.html

[18] National Library of New Zealand. Metadata Standards Framework - Preservation Metadata (Revised). Technical Papers Jun 2003. http://www.natlib.govt.nz/downloads/metaschema-revised.pdf

[19] The OCLC/RLG Working Group on Preservation Metadata (2002). Preservation Metadata and the OAIS Information Mode. A Metadata Framework to Support the Preservation of Digital Objects. June 2002. http://www.oclc.org/research/projects/pmwg/pm_framework.pdf

[20] PREMIS Editorial Committee (2008). PREMIS Data Dictionary for Preservation Metadata, Version 2. March 2008 http://www.loc.gov/standards/premis/v2/premis-2-0.pdf

[21] Thaller, M. et alii (2008). Significant Characteristics to Abstract Content: Long Term Preservation of Content. Springer Lecture Notes in Computer Science (LNCS, vol 5173)

[22] UDFR http://www.udfr.org/

[23] Warmer, A. and Kleppe, A. (2003). The Object Constraint Language. Getting Your Models Ready for MDA. Addison-Wesley Longman Publishing Co., Boston, MA, USA