The International Journal of Digital Curation
Issue 1, Volume 6 | 2011

Implementing Metadata that Guide Digital Preservation Services

Angela Dappert, Adam Farquhar,
The British Library

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 these metadata and develops a data dictionary to support them.[1]

[1] This paper is based on the paper given by the authors at iPRES 2009; received January 2010, published March 2011.

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. ISSN: 1746-8256 The IJDC is published by UKOLN at the University of Bath and is a publication of the Digital Curation Centre.

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 these metadata. From this it develops a data dictionary to support them.

Related Work

Digital preservation metadata are 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 (2006) and Lavoie and Gartner (2005). In 2002, the Reference Model for an Open Archival Information System (OAIS; Consultative Committee for Space Data System (CCSDS), 2002) provided a framework to unify the concepts and terminology in the community. Its information model (Online Computer Library Center/Research Libraries Group (OCLC/RLG), 2002) defines categories for preservation metadata. In 2005 the Preservation Metadata Implementation Strategies (PREMIS) data dictionary[2] consolidated several earlier efforts (e.g., CURL Exemplars in Digital Archives (Cedars) Project[3]; Lupovici & Masanès, 2000; National Library of Australia, 1999; National Library of New Zealand, 2003) to produce conceptual models and concrete metadata dictionaries for implementers of digital preservation services. Now in its second version (PREMIS, 2008), it has been widely accepted and plays a key role in creating coherence in the digital preservation metadata community. PREMIS provide 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.

Contributions

This paper draws on the practical experience gained in the project: Preservation and Long-term Access through Networked Services (Planets;[4] Farquhar & Hockx-Yu, 2007), 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 (Dappert, Ballaux, Mayr & van Bussel, 2008; Dappert & Farquhar, 2009a). The model

[2] PREMIS data dictionary for preservation metadata (Version 1): http://www.oclc.org/research/projects/pmwg/premis-final.pdf.
[3] Cedars Project: http://www.webarchive.org.uk/wayback/archive/20050410120000/http://www.leeds.ac.uk/cedars/index.html.
[4] Planets: http://www.planets-project.eu/.

has consequences for implementations of preservation metadata dictionaries, property registries, and preservation services.

Properties, Values, Characteristics, and Requirements

In order to write with a reasonable level of precision, we need to introduce a basic vocabulary (Chaudhri, Farquhar, Fikes, Karp, & Rice, 1998):

- **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`[5] 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`.

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 (Dappert et al., 2008; Dappert & Farquhar, 2009a, 2009b) 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

[5] An additional typeface (e.g. `file`) is used in this paper to indicate descriptions of operations and files used by the authors' computer program.

`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 characterize 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`.

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.

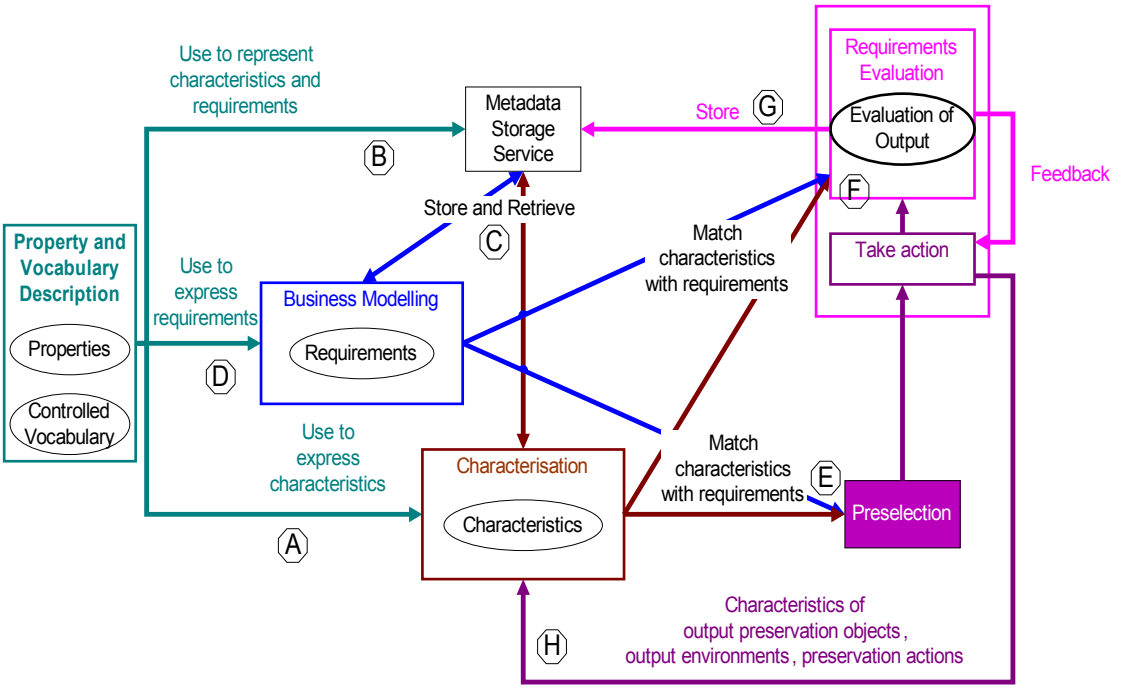

Figure 1. Interaction of Properties, Characteristics, and Requirements.

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[6] or Unified Digital Format Registry (UDFR)[7], can associate file formats with their applicable properties.

[6] The National Archives, PRONOM: http://www.nationalarchives.gov.uk/pronom/.
[7] UDFR: http://www.udfr.org/.

Characteristics extraction languages, such as XCEL: eXtensible Characteristic Extraction Language (Thaller, Heydegger, Schnasse, Beyl, & Chudobkaite, 2008), additionally describe how values for these properties can be extracted from files in a given format. Preservation metadata dictionaries, such as PREMIS (2008), 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[8], 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[9] file (Figure 1A, Figure 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 (Figure 1A, Figure 3);
- Enable metadata storage services to refer to properties unambiguously and to 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).

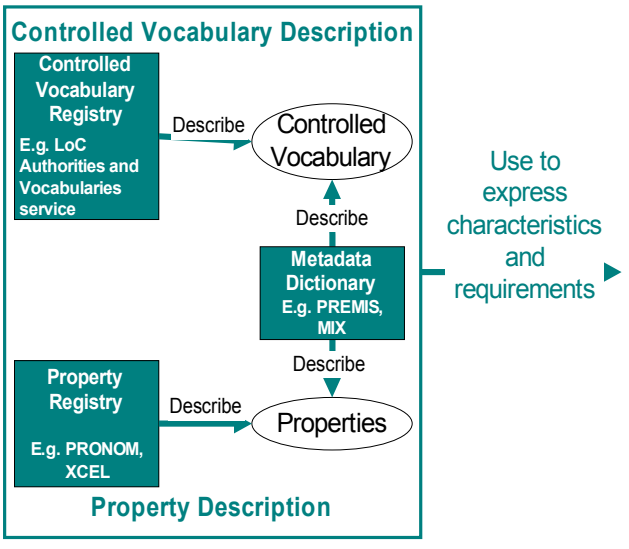

Figure 2. Properties, their Descriptions and their Permissible Values Captured by Controlled Vocabulary Registries. LoC = Library of Congress; MIX = Metadata for Images in XML Standard[10].

[8] The Library of Congress Authorities and Vocabularies service: http://id.loc.gov/authorities/about.html.
[9] 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.
[10] MIX: http://www.loc.gov/standards/mix/.

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 (Journal Storage /Harvard Object Validation Environment (JSTOR/JHOVE)[11]; Digital Record Object Identification (DROID)[12]; Thaller et al., 2008) or assigned manually (Figure 3);
- Characteristics of preservation services can be determined experimentally in preservation testbeds (e.g., Aitken et al., 2008; Figure 3);
- Characteristics may be stored in metadata storage services or produced on demand (Figure 1C).

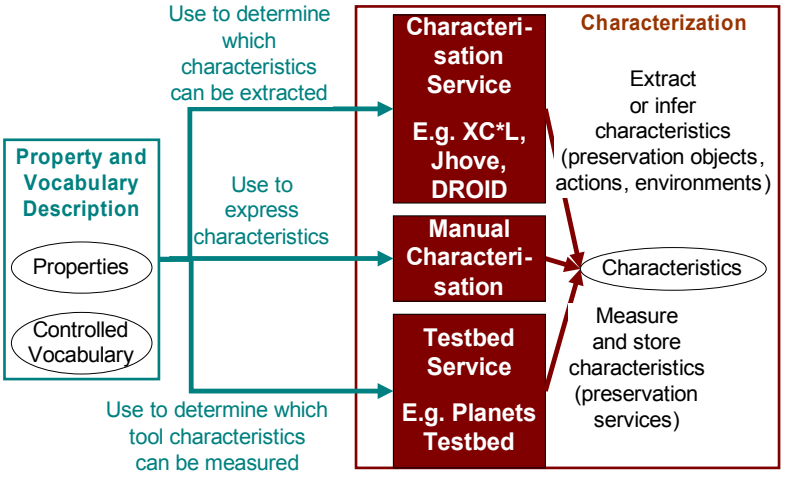

Figure 3. Characterization Service Determination of Preservation Object Characteristics.

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.

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 document the constraints that have been, or should be, applied to specific preservation objects (see Figure 1C). The PREMIS data dictionary for preservation metadata (2008) 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).

[11] JHOVE: http://hul.harvard.edu/jhove/.
[12] The National Archives, DROID: http://replay.waybackmachine.org/20090606081919/http://droid.sourceforge.net/wiki/index.php/Introduction.

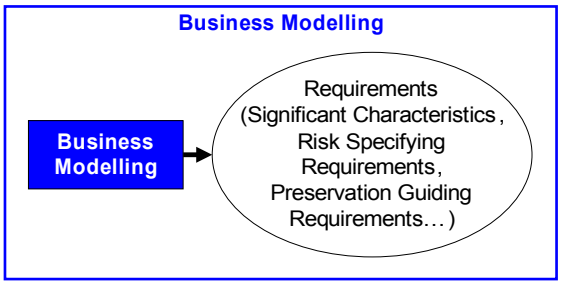

Figure 4. Formulation of Requirements as a Result of Business Modelling.

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 have been obtained in testbed services, is particularly helpful in this step.

Primarily, requirements guide actions, such as preservation monitoring, preservation planning, and preservation execution services (see Figure 5).

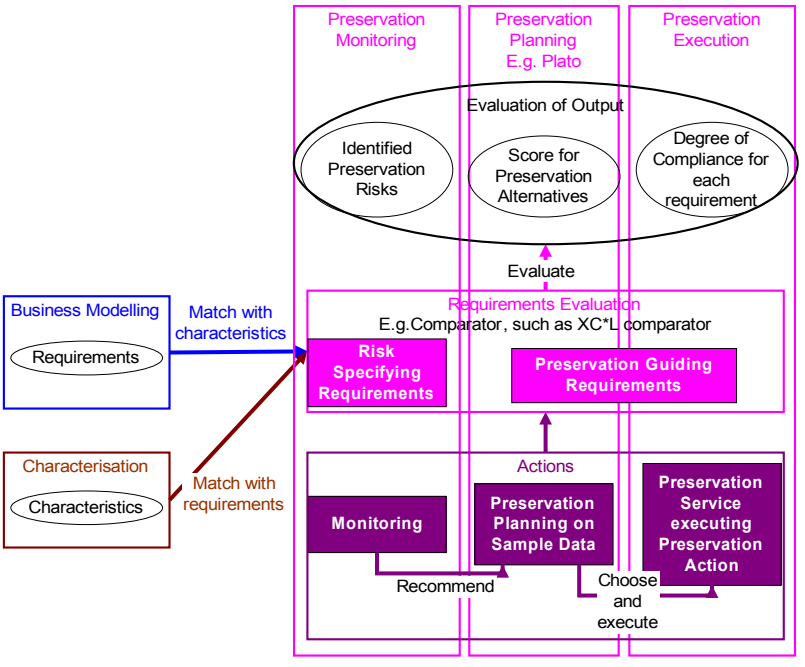

Figure 5. Uses of Characteristics and Requirements.

Preservation monitoring services determine whether risk-specifying requirements are violated, and, therefore, whether 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 (Becker, Kulovits, Rauber, & Hofman, 2008), 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 eXtensible Characteristic Definition Language (XCDL) comparator (Thaller et al., 2008), 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, Figure 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 cannot 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.

Some Observations

Observations for Properties

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[13]. 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.

[13] The association of properties with digital object types is discussed in the Planets testbed (Helwig, 2007). We are refining this to include the type of a component of the digital object, since a logical object might well contain, for example, text, sound, and image components together.

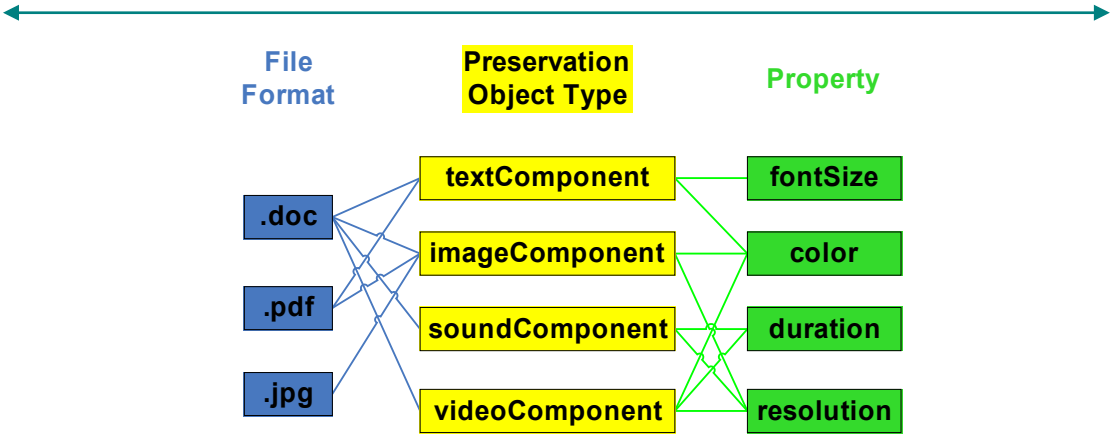

Figure 6. Mapping of Applicable Properties to Formats via Preservation Object Type.

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.

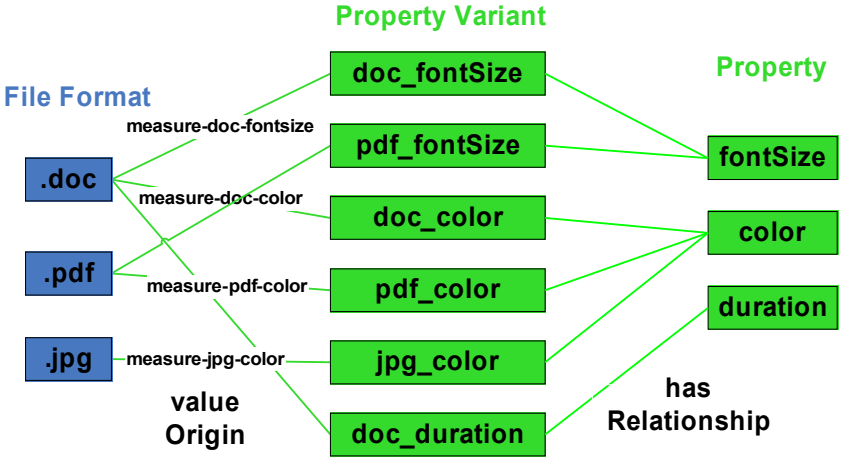

Figure 7. Value Origins and Relationships Between Properties.

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, de-commissioning 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.
    - The `contentType` of objects in an eJournal ingest system is always set to "eJournal" upon ingest.
    - The `budget` of an institution may be set during the execution of a preservation action as follows: `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. For example:
    - `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.
    - `MIMEtype` 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.
    - The `aspectRatio` of an image may be `imageWidth` / `imageHeight`.

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[14]. 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.

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

Observations for Requirements

Observation 8. In many cases, a stakeholder may express requirements dependent on additional conditions, for example, 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 Object Constraint Language (OCL; Warmer & Kleppe, 2003) 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.

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 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.

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: `PropertyID`).
- `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. (2008) 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: `PropertyID`).
    - `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: `EventID`).

Value Origin

The `ValueOrigin` concept provides a way to specify where a particular `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, measured 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: `UnitID`).
- `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: `AgentID`).
- `hasTrigger` (0...n): a trigger for `Value` assignment, for example: 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, for example: 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: `AgentID`). There may be multiple possible agents.

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): specifies for which `Property` the `Characteristic`'s `Value` holds (data constraint: `PropertyID`).
    - `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.

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: `EventID`).

The `requirementSpecification` can be expressed informally or implemented using a constraint language such as OCL (Warmer & Kleppe, 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`.

Conclusions

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 implementation 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 (Dappert et al., 2008; Dappert & Farquhar, 2009a).

Acknowledgments

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

References

Aitken, B., Helwig, P., Jackson, A., Lindley, A., Nicchiarelli, E., & Ross, S. (2008). The Planets testbed: Science for digital preservation. *Code4Lib Journal, 3*. ISSN 1940-5758.

Becker, C., Kulovits, H., Rauber, A., & Hofman H. (2008). Plato: A service oriented decision support system for preservation planning. *In Proceedings of the 8th ACM/IEEE-CS Joint Conference on Digital Libraries.* Pittsburgh, PA.

Caplan, P. (2006). *Preservation Metadata.* In Digital Curation Manual (Version 1.0). Digital Curation Centre: Warwick, UK. Retrieved March 1, 2011, from http://www.dcc.ac.uk/sites/default/files/documents/resource/curation-manual/chapters/preservation-metadata/preservation-metadata.pdf.

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.* Menlo Park, CA: American Association for Artificial Intelligence

Consultative Committee for Space Data Systems (2002). *Reference model for an open archival information system.* ISO: 14721. Retrieved November 24, 2009, from http://public.ccsds.org/publications/archive/650x0b1.pdf.

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. Retrieved November 24, 2009, from http://www.planets-project.eu/docs/reports/Planets_PP2_D2_ReportOnPolicyAndStrategyModelsM24_Ext.pdf.

Dappert, A., & Farquhar, A. (2009a). Modelling organizational preservation goals to guide digital preservation. *International Journal of Digital Curation 4, (2).* Retrieved November 24, 2009, from http://www.ijdc.net/index.php/ijdc/article/viewFile/123/103.

Dappert, A., & Farquhar, A. (2009b). Significance is in the eye of the stakeholder. *13th European Conference on Digital Libraries.* Berlin: Springer. Retrieved November 24, 2009, from http://www.planets-project.eu/docs/papers/Dappert_Significant_Characteristics_ECDL2009.pdf.

Farquhar, A., & Hockx-Yu, H. (2007, November). Planets: Integrated services for digital preservation. *International Journal of Digital Curation 2, (2)*.

Helwig, P. (2007). *Test methods for testbed.* Planets report TB/3-D2. Planets. Retrieved November 24, 2009, from http://www.planets-project.eu/docs/reports/Planets_TB3-D2_MethodsForTesting.pdf.

Lavoie, B., & Gartner, R. (2005). *Preservation metadata.* Technology Watch Report No. 05-01. Digital Preservation Coalition. Retrieved November 24, 2009, from http://www.dpconline.org/docs/reports/dpctw05-01.pdf.

Lupovici, C., & Masanès, J. (2000). *Metadata for long term preservation.* Report Series 2: Networked European Deposit Library. Retrieved November 24, 2009, from http://nedlib.kb.nl/results/NEDLIBmetadata.pdf.

National Library of Australia. (1999). *Preservation metadata for digital collections*. Retrieved November 24, 2009, from http://www.nla.gov.au/preserve/pmeta.html.

National Library of New Zealand. (2003). *Metadata standards framework - preservation metadata* (Revised). Retrieved November 24, 2009, from http://www.natlib.govt.nz/downloads/metaschema-revised.pdf.

Online Computer Library Center/Research Libraries Group Working Group on Preservation Metadata. (2002). *Preservation metadata and the OAIS information model: A metadata framework to support the preservation of digital objects*. Retrieved November 24, 2009, from http://www.oclc.org/research/activities/past/orprojects/pmwg/pm_framework.pdf.

Preservation Metadata Implementation Strategies Editorial Committee. (2008). *PREMIS data dictionary for preservation metadata* (Version 2). Retrieved November 24, 2009, from http://www.loc.gov/standards/premis/v2/premis-2-0.pdf.

Thaller, M., Heydegger, V., Schnasse, J., Beyl, S., & Chudobkaite E. (2008). Significant characteristics to abstract content: Long term preservation of information. In Christensen-Dalsgaard, B., Castelli, D., Jurik, B., & Lippincott, J., (Eds.) *Lecture Notes in Computer Science: Vol 5173. Research and Advanced Technology for Digital Libraries (p. 41-49).* Berlin: Springer.

Warmer, A. & Kleppe, A. (2003). *The object constraint language. Getting your models ready for MDA*. Boston: Addison-Wesley Longman.