Yannis Marketakis

# Preservation Information Conceptual Model

slide 1 In this statement, Yannis Marketakis presents the Preservation Information Conceptual Model and the related knowledge management issues.

slide 2 After an introduction, some aspects concerning preservation are identified; the presentation then focuses its attention on two issues, the preservation of intelligibility and the preservation of provenance, and on one of the CASPAR key components, i.e. the Knowledge Manager. Finally, the data which is currently available followed by the current deployments of the software components are shown.
At the end of the presentation, there is a video demonstration regarding the GapManager.

slide 3 In the digital preservation context, there are some important questions concerning:

- What digital information preservation is;
- What kind and how much Representation Information (RepInfo) we need, and how this depends on the Designated Community;
- What kind of automation we could offer.

The contribution of CASPAR consists of a formalization of intelligibility and intelligibility gap through the notion of dependency. This approach can provide some answers to the above issues.

slide 4 Two examples, the Phaistos disk and the pyramids, bear evidence to the fact that not always do we have the ability to understand the meaning of the resources which we preserve or to reconstruct the processes of their creation.

slide 5 Analogous problems are related to digital preservation which requires the certainty of the capacity to understand a bits stream in the future and also the ability to preserve the intelligibility and the provenance of a resource, e.g. for a satellite image it is necessary to maintain information regarding the way in which it was originally derived, when and by whom it was taken and how it has been processed.

slide 6 The example above makes it evident that not only do we need to preserve the bits, but also the information carried by digital objects and what is necessary for their accessibility, integrity, authenticity provenance and intelligibility by human or artificial actors.

slide 7 Concerning the intelligibility, the RepInfo is one of the key concepts which are offered by the OAIS conceptual model.
An Information Object needs RepInfo to be understood; in turn, the RepInfo is an Information Object itself and requires other ReInfo. A way to stop this potentially long chain of RepInfo is necessary.

slide 8 In order to abstract from the various domain-specific details, a general module has been adopted; it consists of two main concepts: the notion of module and that of dependency.
A module could be a piece of software/hardware or a knowledge model, formally expressed (e.g. an ontology) or implicitly expressed.
The notion of dependency says that a module t depends on t', if t requires t' in order to be understood, managed and so on.
So, the RepInfo requirements of OAIS have been modeled as dependencies between modules.

slide 9 Intelligibility can be formalized based on this notion.
For instance:

- to understand a file named README.txt which contains an English test, a text editor is needed in order to open it, but also the knowledge of the English language is necessary;
- a multimedia performance data depends on the understandability of the C3D motion files, DirectX and MAX/MSP modules;
- to understand a FITS file, it is necessary to understand the FITS standard, the FITS dictionary and so on.

slide 10 These patterns occur in formal expressed knowledge, for instance in the semantic web or in a schema which may extend or use several ontologies or other schemas.
It is also possible to express them in a dependency graph over namespaces, where we can have a chain of dependencies between different namespaces that end in the RDF standard.

slide 11 One question which arises is: how many dependencies may we have?
A systematic way is needed in order to limit the dependency chain.
To do it, the notion of Community Knowledge has been introduced which, according to OAIS, aims at preserving digital information for a particular community.
The knowledge that is already available to the members of a community may be formalized in order to obtain a Designated Community Profile, which is just a set of modules that are assumed to be known by that community. This means that there is

no need to analyze the dependence of these modules since each member of the community already knows them.
(In the slide an example shows some modules that depend on each other, and that there is a community, identified as Tu, which knows the module t3 and t6).

slide 12 There are also various auxiliary notions:

- the notion of closure, i.e. a set of full direct and indirect dependencies of one module (in the slide it is possible to see the closure of module tx and of module ty, which is also the closure of profile tu).
- the notion of intelligibility gap, i.e. the smallest set of extra modules that the identified community u needs in order to understand the module t.

slide 13 (The example in the slide shows that the gap between the module ty and the profile u is empty since the profile u already knows the dependencies of module ty; while the gap between module tx and profile u is the set of modules t1, t2, t4 and t5).
This means that in order to preserve a digital object t for a community with a profile Tu, it is necessary to obtain and store only the gap of module t and profile u, since the rest is already known.
Instead, in delivering an object t to an actor with a profile u, the extra modules that constitute the set gap between module t and profile u have to be delivered to him in order to return something intelligible.

slide 14 The notion of intelligibility can be exploited in archiving; in fact digital objects are archived with some extra information and the notion of intelligibility is exploited in order to identify the extra modules to archive.
(The example in the slide shows three different types of Archival Information Packages (AIP) of the same object o1 for three different communities).

slide 15-16 (The slide shows a symbolic example concerning module dependencies and Designated Community Profiles; more specifically, there are three digital objects: o1, which is a PDF document; o2, that is a FITS file; and o3, which is a ZIP file that contains multimedia performance data. Furthermore, three Designated Community Profiles have been designed: P1, which is used for astronomers; P2, that identifies casual users and P3, which is used for multimedia users. The gaps between the objects and the related profiles is made clear too).

slide 17 (This is an example from cultural testbed that presents an ASCII grid file and its dependencies).

slide 18 These theories have been implemented by adopting semantic web languages for modeling modules and dependencies, in this way they obtain advantages in terms of

extensibility and inheritance. There is a minimal top-level ontology where profiles know some modules that depend on each other; moreover, there is a typology of modules which can be extended as required.

slide 19 Another question which arises is: is there a clear way to identify which the dependencies of an object are? Actually, to determine the dependencies is an objective.
(In the example: if we have a a.java file and we want to compile it, its dependencies include a java compiler; on the other hand, if the aim is to edit and read it, the file would just depend on the ASCII format; finally if we want to run a program, it would depend on a java virtual machine.
So, there are several dependency types which can be implemented easily using the semantic web languages since the *dependsOn* relation has already been specialized and the notion of intelligibility has been extended accordingly).

slide 20 Now, we are going to explain the issues concerning the preservation of provenance.

slide 21 CIDOC-CRM ontology has been extended in order to be able to document digital objects.
CIDOC-CRM can be considered as a backbone to represent provenance information.
On the architectural models there is the CIDOC-CRM on the top and below there are several domain specific specializations of it; the metadata can be expressed as instantiation of the above schema.

slide 22 (An example concerning the transfer of custody of information objects i.e. a painting of Van Gogh shows the different owners of this object).

slide 23 (Here is how the derivation history can be represented; specifically there is an image which is converted from JPEG format to PNG format, and another PNG image which a smaller resolution which was derived from the initial PNG image).

slide 24 The Knowledge Manager is a type of software that permits one to deal with all these tasks.

slide 25 It comprises two layers: a Semantic web Knowledge Middleware (SWKM) which is the lower layer providing a set of core services for managing Semantic Web data (storing, updating and so on.) and on top of this, there is a second layer, the Gap Manager, which is responsible for the dependency management and for the intelligibility gap.

slide 26 A picture shows the components of the CASPAR project and where the Knowledge Manager is located.

slide 27 Concerning the available data, a Core ontology for the GapManager has been identified and information has been exported regarding several formats from existing registries, like PRONOM, from various tesbeds and from the registry of CCLRC; in addition to which, an ontology for provenance has been developed.

slide 28 Regarding current deployments, there is one repository in Pisa which is used by Finding Aids, another in Crete, in FORTH-ICS from which all the information about the exporting data formats has been kept, and there are also various other deployments.

slide 29 In brief, Gap Manager can aid the following tasks:

- the decision of what metadata needs to be captured and stored;
- the identification of the data objects that are in danger in case a module (e.g. a software component or format) is becoming obsolete (or has already vanished);
- the reduction of the metadata that has to be archived, or delivered to the users.

slide 30 The presentation above is followed by a demonstration concerning the Gap Manager.

demo example 1 In order to preserve a single digital file, a package containing all the dependencies of the modules required to made the file intelligible should be created (this is what has been called ‘closure’).
To preserve, the logo of CASPAR, for example, (a module whose identity is a file) it is possible to access it on the web by a URL, add a name and a version of the module, specify that its intelligibility depends on the availability of JPEG format; then, after selecting the new module and asking for its closure, it is possible to visualize its direct dependencies and its closure; so that these may be inserted into a package that could be expressed in different packaging formats.
Next, by choosing one option from a list of different profiles, it is possible to see how the knowledge that a Designate Community is supposed to have, determines the contents of the package.
By selecting both the object to preserve and the profile of the community, it is possible to ask the software a gap of the intelligibility, i.e. the comparison between the knowledge of the community and the dependencies of the object to preserve. If the result is an empty package, it means that no other information needs to be added to the package since the object is already intelligible; instead, if the result is

that extra modules are required in order to make the resource intelligible to the community, these must be added to the package.

demo example 2

Supposing that a format is becoming obsolete and that the archive can visualize which object depends on it: just by locating the module of the format it is possible to see a list of the modules which are in danger.

demo example 3

In this example we can imagine having a movie and a file with its overview and we want to compress the two files; in order to preserve the new compressed file a new module must be created with its appropriate dependencies, i.e. the movie file, the description file and the module of the format of compression which was used. By locating the modules that have been added, it is possible to know what the respective dependencies and their closures are.

demo example 4

This example shows how dependency types can be used.
A digital object can depend on others for different purposes.
If a module for a file (e.g. a.java) is created, some dependencies may be added to this module (e.g. the java compiler to compile it, and the module of ASCII format that is necessary in order to edit it).
After looking for the modules on which the file a.java depends on, the result will contain both the java compiler and the ASCII modules; but, if one searches for modules on which a.java depends to be edited, the result will contain only the ASCII format module; analogously, changing the dependencies type from *dependsEdit* to *dependsCompile*, then the result will contain only the java compiler module.