ON PROVIDING INTELLIGIBILITY-AWARE PRESERVATION SERVICES FOR DIGITAL OBJECTS

Yannis Tzitzikas

Institute of Computer Science (ICS) Foundation for Research and Technology – Hellas (FORTH)
GR-711 10 Heraklion, Crete, Greece
Email: tzitzik@ics.forth.gr

ABSTRACT

Preserving digital objects requires preservation of not only their bit-level representation but also their intelligibility. To this end a digital object should be associated with metadata appropriate for interpreting that object; such metadata are often referred as representation information. Even such metadata may not be intelligible, however, so we may have to associate them with extra metadata, and so on. This paper approaches this problem by introducing an abstract model comprising modules and dependencies. Community knowledge is formalized over the same model by introducing the notion of profile. This notion is then exploited for deciding representation information adequacy (during input) and intelligibility (during output). Subsequently some general dependency management services for identifying and filling intelligibility gaps during input and output are described and analysed. Finally a prototype system based on these ideas is described.

Keywords: Digital Preservation, OAIS, Intelligibility, Dependency Management, Semantic Web, Metadata

1 INTRODUCTION

The preservation of digital information is an important requirement of the modern society. Digital information has to be preserved not only against hardware and software technology changes, but also against changes in the knowledge of the community. The statement of Heraclitus "Everything flows, nothing stands still" also characterizes the digital era. According to the OAIS Reference Model (OAIS), metadata are distinguished into various broad categories. One very important (for preservation purposes) category of metadata is named *Representation Information (*RI), which aims at enabling the conversion of a collection of bits to something meaningful and useful. For instance, and according to OAIS, the RI of a digital object should comprise information about the *Structure*, the *Semantics*, and the needed *Algorithms* for interpreting and managing the digital object.

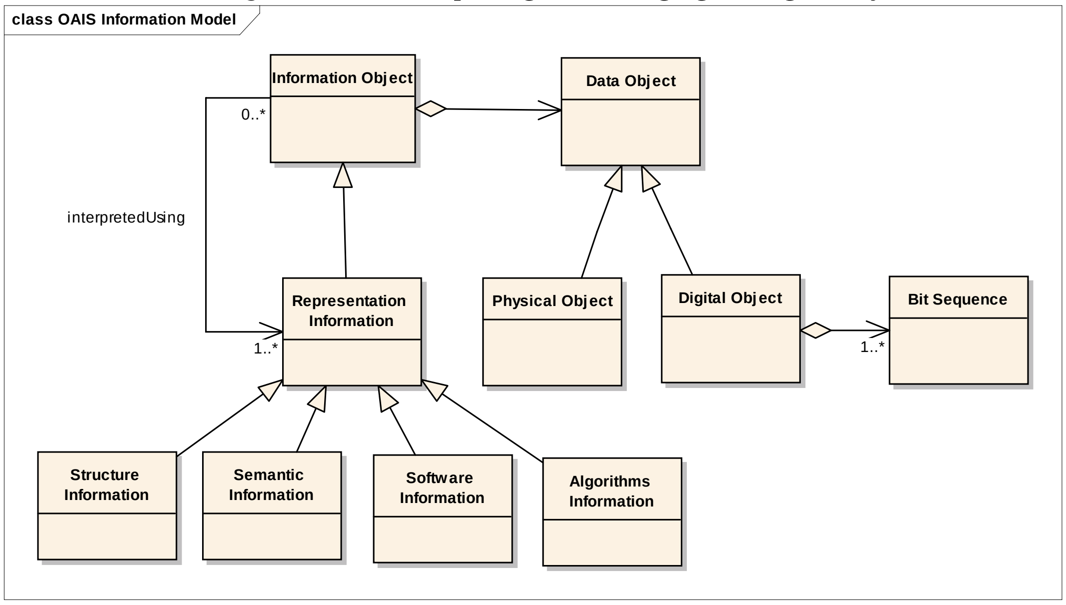


Figure 1. The Information Model of OAISFigure **1** illustrates the main structural entities of the OAIS information model in the form of a class diagram. However even such metadata (i.e. RI) may not be intelligible so we may have to associate them with extra metadata, and soon (notice that in Figure 1, the association *interpretedUsing* is inherited to the class RepresentationInformation). This raises the following important question:

[Q]: How much RI should we create to ensure intelligibility?

To approach this question in a domain-independent manner, we adopt an abstract model comprising *modules* and *dependencies.* In brief, the RI requirements of a digital object are viewed as dependencies. Returning to the question [Q], one important remark is that different users or communities of users (consumers or providers) have different characteristics and knowledge. For instance, although we could make the assumption that all users know what a pdf file is, we cannot make the same assumption for other types of files (e.g. for an .rdf file or for a .FITS file). For this reason we introduce the notion of a Designated Community (DC) *profile.* The profile of one community (or of a single user) actually expresses the RI's that are assumed to be known by that community (or user). With these notions (module, dependency, and DC profile), we can formalize the problem of intelligibility and can address the question *[Q]*. It is also worth mentioning that according to OAIS (OAIS), OAIS is defined as *an archive, consisting of an organization of people and systems that has accepted the responsibility to preserve information and make it available for a Designated Community*. However, the information model of OAIS does not introduce any element corresponding to the notion of DC. From this point of view, we may say that the model that we propose is an extension of the OAIS information model allowing us to express explicitly the assumptions regarding DC knowledge and to subsequently automate some parts of the problem. The intelligibility-awareness processes that we describe constitute an extension of the OAIS functional model.

This work can be exploited for building advanced preservation information systems and registries. This research is being done in the context of the ongoing EU project CASPAR (FP6-2005-IST-033572)[1] whose objective is to build a pioneering framework to support the end-to-end preservation lifecycle for scientific, artistic, and cultural information. This paper actually elaborates on the ideas that were originally described in (Tzitzikas, 2007). This paper contains more specific examples, describes profile-based AIPs and DIPs, discusses various methodological issues, and describes the design of a prototype implementation.

The rest of this paper is organized as follows: Section 2 defines intelligibility through dependencies, while Section 3 discusses descriptive metadata and proposes architecture for such metadata schemas. Section 4 discusses technical aspects and the design of a prototype system. Finally, Section 5 summarizes and identifies directions for further research.

2 MODELING INDELIBILITY THROUGH DEPENDENCIES

In order to abstract from the various domain-specific and time-varying details, we introduce the general notions of Module and Dependency. We adopt a quite broad definition. A module can be a piece of software or hardware, a knowledge model expressed explicitly and formally (e.g. an ontology), or a knowledge model not expressed explicitly (e.g. Greek Language). The only constraint is that modules need to have a unique identity. Concerning dependencies, a module t depends on t', written t>t', if t requires t'. Broadly speaking, the meaning of a dependency t > t' is that t cannot function, be understood, or managed without the existence of t'. Therefore, we model the RI requirements of the OAIS information model as dependencies among modules. Some examples are illustrated in Figure 2.

[1] Work partially supported by European Community under the Information Society Technologies (IST) programme of the 6th FP for RTD - project CASPAR contract IST-033572. The authors are solely responsible for the content of this paper. It does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of data appearing therein.

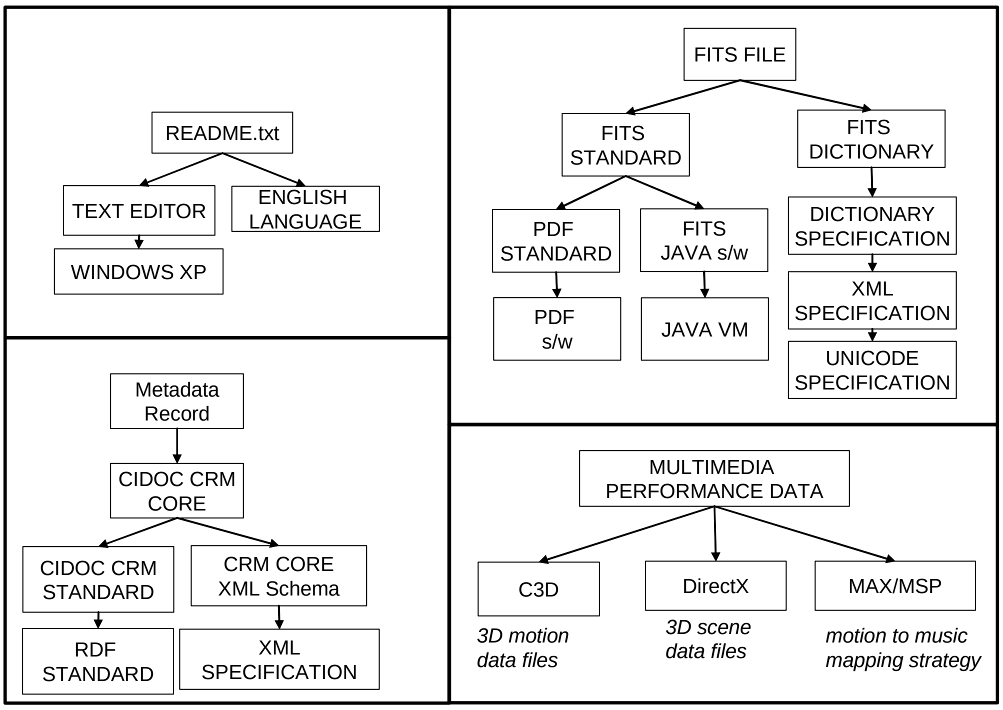


Figure 2. Examples of modules and dependencies

For instance, the intelligibility of a digital object README.txt (upper left box) depends on the availability of text editors and knowledge of the English language. The remaining examples come from the various testbeds of the CASPAR project (e.g. FITS files are used by astronomers). Dependencies also exist among formally expressed knowledge. For instance, the left part of Figure 3 sketches a number of Semantic Web (SW) schemas (recall that a SW schema may reuse or extend elements coming from different schemas), and the right part shows the corresponding dependency graph.

Concerning community knowledge, an actor or community u can be characterized by a ***profile*** Tu that contains those modules that are assumed to be available or known to u (i.e. Tu $\subseteq$T). For example, if u is an artificial agent, then Tu may include the software or hardware modules available to it. If u is a human, then Tu may include modules that correspond to implicit or tacit knowledge. Note that if preservation is done for a particular Designated Community, we may call these *DC profiles*. One can easily guess that what the dependencies really are strongly depends on the DC and on its purposes (this issue in discussed in more detail in Section 2.1).

Now we introduce an assumption, namely the *unique module assumption* (UMA), which is very useful for both theoretical and practical reasons. According to UMA, each module is uniquely identified by its name, and its dependencies are always the same. In practice, we may adopt a more relaxed assumption of the form: different modules have different identities.

If T denotes the set of all modules, then the dependency relation > is actually a binary relation over T. We shall use Nr(t) to denote the direct dependencies of t, i.e. Nr(t)={t' | t > t'}. We shall use $>^+$ to denote the transitive closure of >, and $>^*$ to denote the reflexive and transitive closure of >. Analogously, we can define $Nr^+(t)$ = { t' | t $>^+$ t' } and $Nr^*(t)$ = { t' | t $>^*$ t' }.

In order to formalize the notion of intelligibility, we introduce the notion of *closure*. The closure of a module t is defined as C(t) = $Nr^*(t)$. The closure of a set of modules S (where S $\subseteq$T) is defined as C(S) = $\cup$ {C(t) | t $\in$ S}. As Tu is a set of modules, we can define its closure as C(Tu).

Recall that $Nr^+(t)$ is the set of all dependencies of t, i.e. $Nr^+(t)$=C(t)-{t}. We may denote this by $C^+(t)$, so it is actually the set of all (direct or indirect) dependencies of t. Figure 4 shows a dependency graph and the profile Tu of an actor u.

It follows easily that u can understand a module t if and only if $C^+(t) \subseteq C(Tu)$. In the running example, u can understand ty but he cannot understand tx.

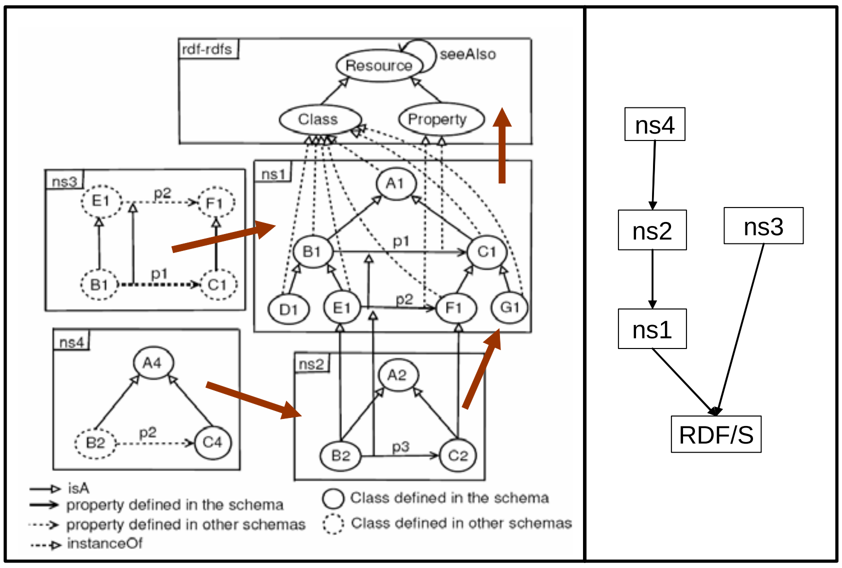


Figure 3. Dependencies between RDF schemas

We can now define the *intelligibility gap* as the smallest set of extra modules that u needs to have in order to understand a module t. We can denote this by **Gap(t,u),** and it follows easily that $Gap(t,u) = C^+(t)-C(Tu)$ where "-" denotes set difference. In our example, $Gap(ty,u)= \emptyset$, while $Gap(tx,u)= \{t1, t2, t4, t5\}$. Notice that due to UMA, it is implied that u can also understand t7 and t8 even if they are not elements of Tu. In addition and due to UMA, we can decide whether a module is intelligible by inspecting only the direct dependencies of t. In particular it holds: $C^+(t) \subseteq C(Tu) \Leftrightarrow \max(C^+(t)) \subseteq C(Tu)$. In our example, $\max(C^+(ty))) = t3 \in C(Tu)$, while $\max(C^+(tx))=t1 \notin C(Tu)$.

According to the previous discussion, an intelligibility gap can be filled by getting the missing modules. This means that if we want to preserve a digital object t for a community with profile Tu, then we need to get and store only Gap(t,u) plus an id that denotes Tu. Analogously, if we want to deliver an object t to an actor with profile Tu, then the only extra modules that we should deliver to him, in order to return him something intelligible, is the set Gap(t,u).

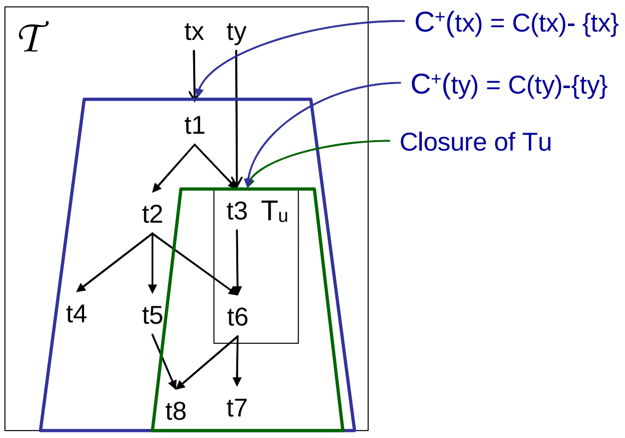


Figure 4. Dependency graph, profile and closure

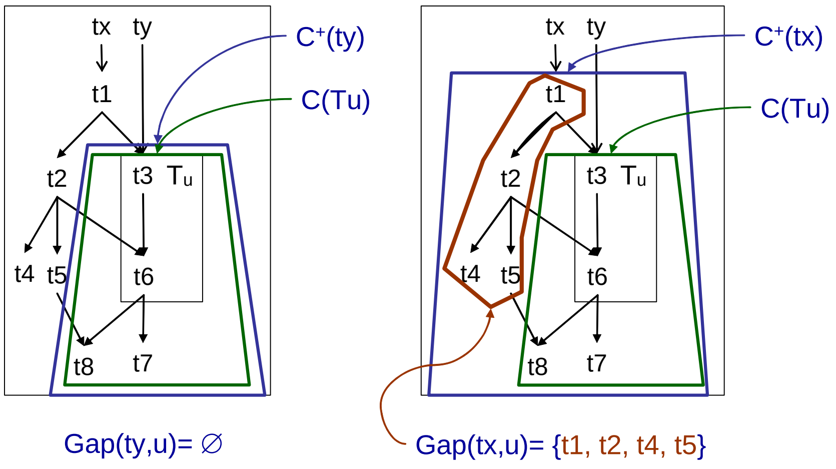


Figure 5. Example of Gaps

To summarize, the above formalism allows us to provide specific answers to the following very important questions:

(a) How much RI should we create?

(b) How this depends on the knowledge of the designated community?

(c) What automation is needed?

Recall that according to the OAIS Reference Model (OAIS), an AIP (Archival Information Package) is actually a format that consists of the Data Object, the required RepInfo, plus PDI (Preservation Description Information). Now a DIP (Dissemination Information Package) is an information package delivered to the Consumer in response to an access request, and it may differ in form (e.g. TIFF to JPEG) or content (e.g. amount of metadata supplied) to that which resides in the archival store. The adoption DC profiles allow defining the "right" AIPs. Specifically we can define AIPs that are intelligible for certain communities and at the same time are redundancy free. The same is true for DIPs. This is illustrated in Figure 6 where for an object o1 three different AIPs are defined (for DC1, DC2, and DC3). The DIPs of o1 for DC1, DC2, and DC3 are actually the corresponding AIPs without the line that indicates the profile.

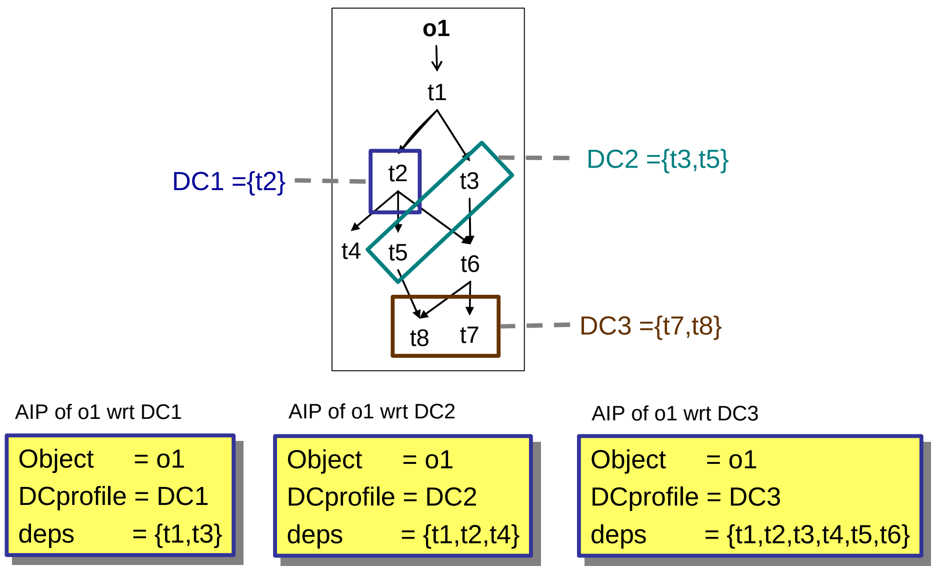


Figure 6. Exploiting DC profiles for defining the "right" AIPs

Of course, the fewer assumptions we make about what the community knowledge is, the more difficult and laborious the problem of recording becomes. With no assumption at all, i.e. if we assume that Tu=∅, then the problem is practically impossible.

However, what about cross-community interpretability? An important remark is that the more explicitly we have represented the knowledge of communities, the more probable cross-community interpretability is. This is illustrated in Figure 7. The left part illustrates 2 digital objects **a** and **b** each depending on one DC profile, A and B respectively. In this case, **a** cannot be understood by community B, and **b** cannot be understood by community A. The right part of the figures illustrates the same situation with the only difference being that these two profiles have been analyzed in more detail. In this case, gaps can be computed, and cross-community intelligibility could be supported. Specifically in that case, we have Gap(**a**,B) = {t1,t3} and Gap(**b**,A)={t2,t5}.

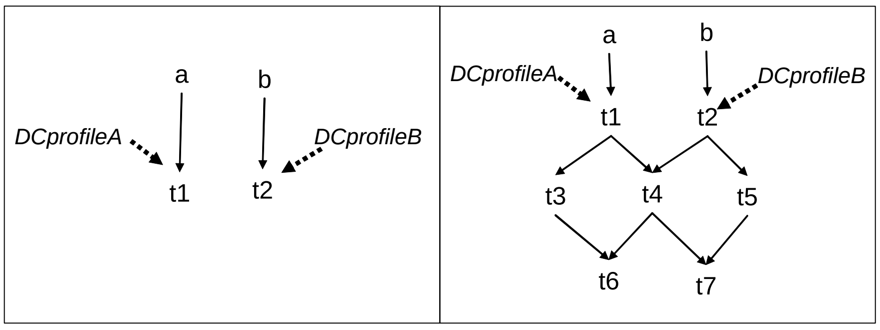


Figure 7. Dependencies and Cross-Community Intelligibility

Community knowledge evolves, however, and consequently, DC profiles may also evolve over time. In that case we can reconstruct the AIPs according to the latest DC profiles. Such an example is illustrated in Figure 8. Notice that the new profile (i.e. DC2') is richer than its previous version. As a consequence, the new AIPs are smaller (in size) than their original version.

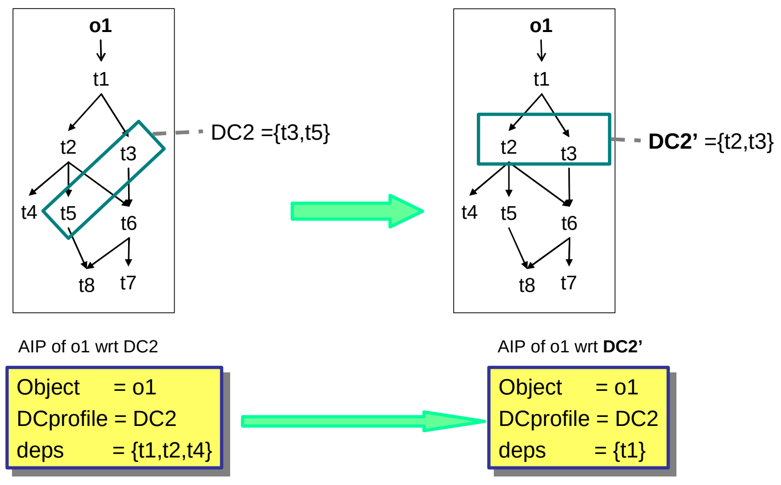


Figure 8. Revising AIPs after DC profile changes

2.1 On defining dependencies

In general, the dependencies of a module strongly depend on what task we may want to perform on that module. For instance, consider a file named a.java containing java source code. If we want to read the source code, its dependency is the ASCII code. If we want to compile it, then its dependencies include the javac compiler; if we want to execute it, then we have to derive the corresponding bytecodes whose execution depends on the availability of a JVM (Java Virtual Machine). From that perspective, we could say that our previous discussions and models (on intelligibility) presuppose that we have decided what tasks we want to perform with a module. Recall that according to the Unique Module Assumption that we adopted earlier, the dependencies of a module are always the same. This means that the dependencies of each module are always defined assuming the same task (or set of tasks). However, from the several kinds of tasks (and hence dependencies) that we might want to perform on a digital object, we need to identify those that are important for the preservation of intelligibility. Therefore, the question is: *With respect to what goal we should record intelligibility dependencies?* Even if we are not able to provide a definitive answer, we should at least provide some guidelines to those (archivists) who are going to extract and record dependencies.

Regarding related work on this subject, we should say that dependencies are ubiquitous, and dependency management is an important requirement that is subject of current research in several (old and new emerged) areas: from Software Engineering (Vieira, 2001; Vieira, 2002; Walter, 2001; Franch, 2003; Belguidoum, 2007), to Ontology Engineering (Jarrar, 2002; Sunagawa, 2003).

Specifically, in software engineering the various build tools (e.g. make, gnumake,nmake, jam, ant) are definitely related (they allow defining dependencies and those tasks required to be performed in order to build a software project). In ontology engineering, an analogous problem is how to reflect a change of an ontology to the dependent ontologies (i.e. to those that reuse or extend parts of it), which may be stored in different sites. Another related problem is that of schema evolution, i.e. the problem of reflecting schema changes to the underlying instances. Actually this problem is related to the evolution of modules and dependencies. Table 1 describes in brief a number of dependency management approaches that have been described in the literature.

Work	Modules	Assumed Goal(when recording dependencies)	Types of Dependencies (between modules)	Reason why dependencies are recorded
[Belguidoum, 2007]	Software components	To install or to uninstall a composite component.	Mandatory, optional, negative.	To reason on installability, deinstallability.
[Franch, 2003]	Software components	Achieve goals, satisfy soft goals, complete tasks, provide and consume resources	Goal, task, resource, soft goal	To aid the selection of the most appropriate component
[Vieira, 2002]	Software components	One goal is the ability to compile/run; another is to express what component *affects* the behaviour of other components.	(a) Internal (i.e. intra-component), and (b) External (inter-component). The former are further categorized to (a1) implementation-based and (a2) operation-based. The external ones are distinguished to (b1) hardware, (b2) software (i.e. required interfaces), and (b3) causal.	To support the process of evolution and testing in component-based systems. The paper presents a preliminary analysis – no specific technique is described.
[Sunagawa, 2003]	Ontologies	No specific goal is mentioned. It considers the dependencies already recorded in the ontology representations (reuse/extend inter-ontology relationships).	Isa, Reference	To aid the development of ontologies in particular when changes occur, i.e. to address questions of the from: if an ontology changes what should happen in the dependent ontologies?

Table 1. Dependency management in other domains

As we can see, there is much heterogeneity on the types of modules, the kinds of goals (that determine what a dependency is), the types of dependencies, and on the dependency management services. Probably in each preservation application domain, we have to model the corresponding modules and dependency types and identify the needed services. It does not seem that we could have a general solution for all kinds of modules and dependencies and their semantics. For this reason, and at least for the structural model, the employment of Semantic Web languages is a promising approach as it offers us the flexibility to extend the typology of modules and dependencies in a straightforward manner. For instance, different goals can be represented by specializing the *dependsOn* relation (i.e. by defining a "subproperty" in the terminology of RDF).

3 ON DESCRIPTIVE METADATA

A preservation information system should also be able to manage descriptive metadata. Although the forms that descriptive metadata can have can not be restricted, the adoption of a general methodology based on international standards is a promising approach. For instance, CIDOC CRM (CIDOC) is a reference ontology (currently an ISO standard 21127) that could be exploited for this purpose. The CIDOC Conceptual Reference Model is a core formal ontology describing the underlying semantics of data schemata and structures from all museum disciplines and archives. It is the result of long-term interdisciplinary work and agreement. It has been derived by integrating (in a bottom-up manner) hundreds of metadata schemas, and its core model is stable (almost no change the last 10 years). In essence, it is a generic model of recording "what has happened" in human scale, i.e. a class of discourse. It can generate huge, meaningful networks of knowledge by a simple abstraction: history as meetings of people, things, and information. Currently, its extension for scientific data is being investigated. FRBRoo is another (under development) ontology that is going to specialize CIDOC CRM. Ontologies, such as CIDOC CRM, FRBRoo, can be exploited for defining domain-specific schemas.

It is worth mentioning that these formal ontologies or metadata schemas can be directly expressed as Semantic Web (SW) ontologies. The later could then be used for describing the objects of interest. The benefit of this approach is that it can alleviate the effort required for defining a domain specific ontology or schema and that the existence of a general upper level promises interoperability. Figure 9 illustrates such architecture of Semantic Web schemas. In particular, CIDOC CRM can be modeled as a SW namespace and FRBRoo as another namespace that specializes it. Under these two schemas, we foresee other specializations for capturing domain-specific requirements.

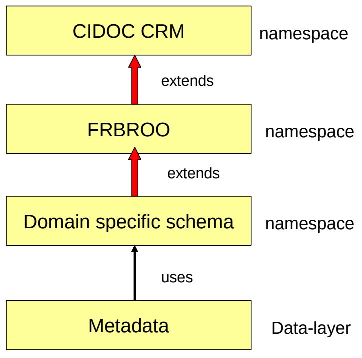


Figure 9. An Architecture of Conceptual Models

Over a knowledge repository that contains such schemas, one could define a plethora of high level services. For instance, there is the ubiquitous requirement for provenance information. Recall the provenance is a type of metadata that should be contained in the PDI (Preservation Description Information) of an information object. A set of provenance-related queries could be designed assuming the CIDOC CRM schema. These queries will return useful information even if applied upon objects that are not described using the CIDOC CRM schema but are a specialization of it. This is an important benefit of adopting SW technologies and the above architecture of models.

It follows that CIDOC CRM could offer cross-community intelligibility and interoperability for descriptive metadata. Figure 10 illustrates the idea. On the left side we have two schemas (named IRCAM and UNESCO), which do not share any common modeling entity, while on the right side we view the same schemas but this time each defined as a specialization of CIDOC CRM. The latter approach allows interoperability and cross-community interpretability.

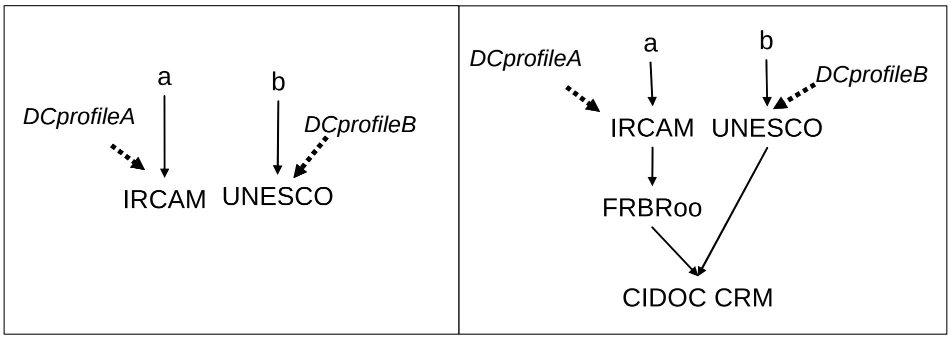


Figure 10. Cross-Community Interpretability

4 TECHNICAL ARCHITECTURE

A Preservation Information System could adopt the notion of profile in order to support intelligibility-awareness services. For instance, it could adopt the following policies: (a) the input (e.g. data objects to be archived) should be intelligible by the system and (b) the output (e.g. returned answers) should be intelligible by the recipients. The notion of profile could be used as gnomon in these policies. Figure 11 illustrates some basic steps of these processes. They include the steps of selecting a profile, identifying the gap, and filling the gap. Moreover, the impacts of changes have to be identified, and the involved parties should be notified. The latter is part of the ongoing curation process. The impacts of changes on the modules and their dependencies are discussed in Tzitzikas and Flouris (2007). We should remark that these processes correspond to the elements of the functional model of OAIS.

Another important remark is that identification and completion of an intelligibility gap could be done either in one step or gradually. The former is in many cases difficult. For instance, consider the example of Figure 3. To compute the closure of n4, one has to be able to parse an RDF file; otherwise one can deduce only ns4 > RDF/S (from the extension of the file). However, a gradual approach would allow one to first fill the gap of RDF/S (e.g. to obtain an RDF/S parser). After this step, he will be able to extract the dependency ns4> ns2 and then try to obtain ns2.

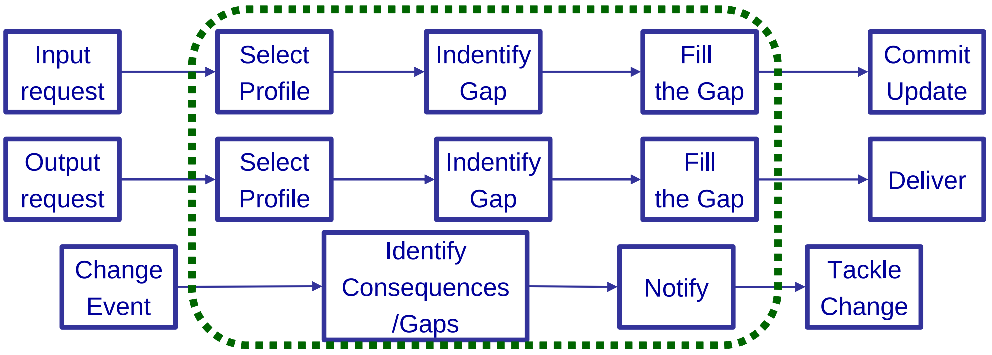


Figure 11. Intelligibility-aware services

There are several options for implementing the above framework and the related services. One approach is to adopt Semantic Web technologies. Figure 12 illustrates a possible architecture of SW schemas and data. The upper level comprises three small SW schemas. The basic dependency management services could need to know only these schemas. The bottom layer shows an indicative instantiation of the above schemas. Between these two layers, a number of other schemas can be defined that specialize or refine the elements of the upper schemas. For instance, the notion of module can be specialized (we could have a typology or taxonomy of modules). Furthermore, the property class *interpretedUsing* can be specialized (the new specialized property class could have as domain and range subclasses of Module). One benefit of adopting Semantic Web technologies and an architecture of schemas such as this is that we can build a preservation system that needs to know only the upper schemas. To be more specific, this means that the queries may be formulated in terms of these schemas. However the same queries function correctly even if the data level instantiates specializations of the above schemas. This is due to the semantics of specialization.

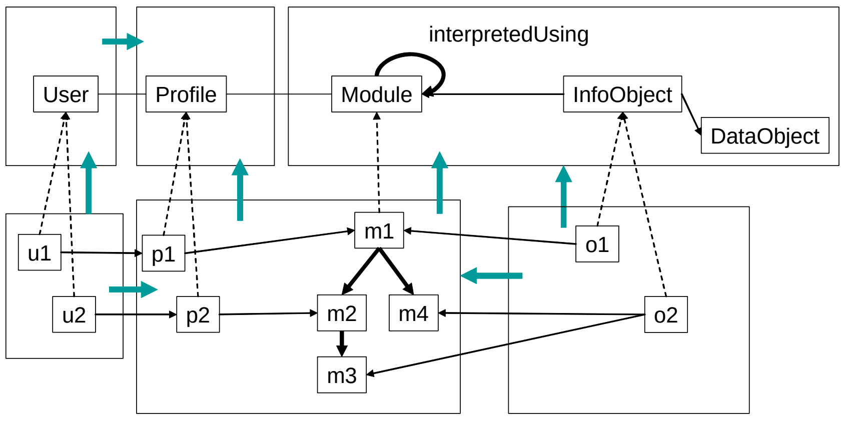


Figure 12. Architecture of Semantic Web Models

A Knowledge Manager component could keep stored the dependency graph and the profiles, while a Preservation Data Store could keep the AIPs (as shown in Figure 13). The Knowledge Manager could provide object DIPs according to DC Profiles different that those that have been used for archiving the packages.

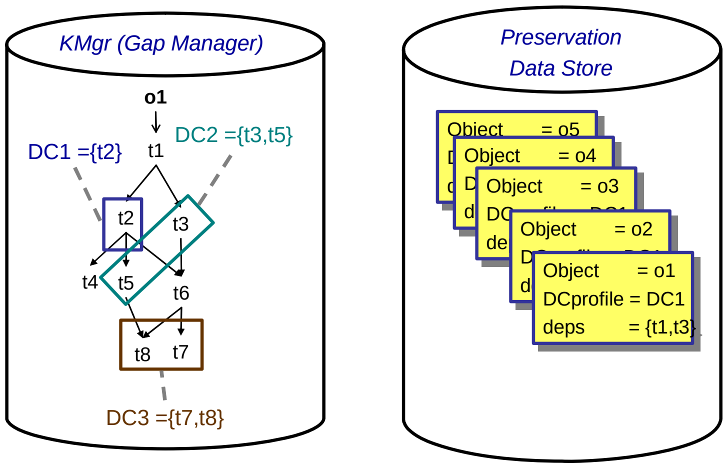
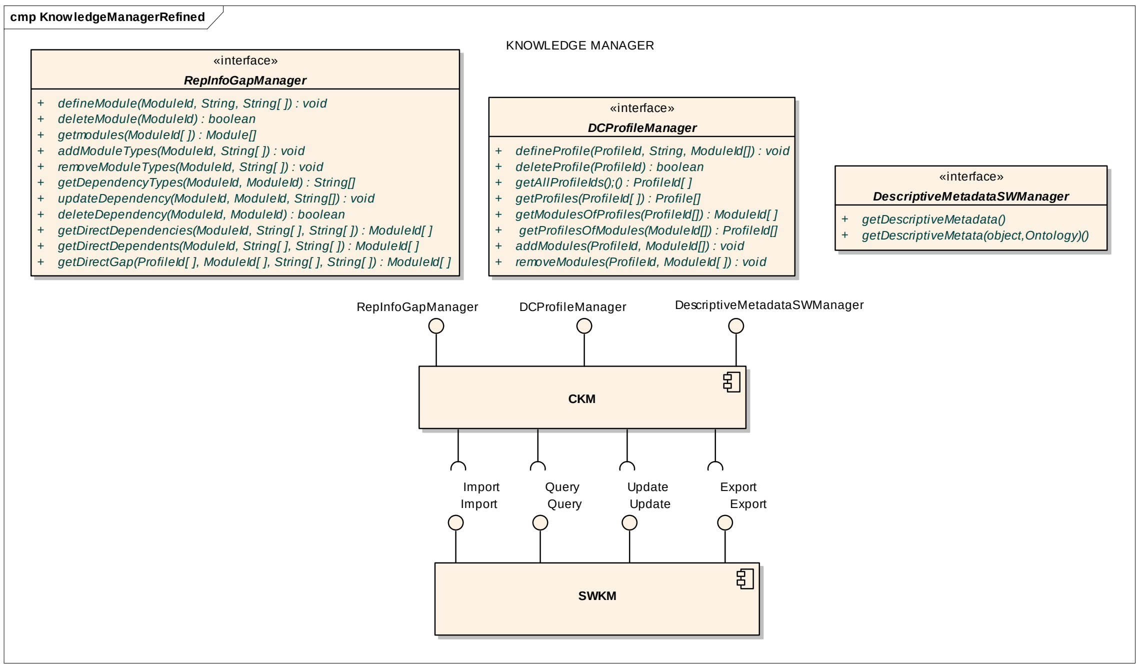


Figure 13. Knowledge Manager and Preservation Data Stores

We are currently developing a proof-of-concept prototype able to manage DC profiles, modules, and dependencies (converters and emulation are out of the scope of this prototype). The prototype has a modular design that enables changing the persistence layer easily. The current implementation has two persistence storage managers: one plain file-system based and another one over a semantic web repository, specifically over SWKM (Semantic Web Knowledge Middleware). Figure 14 illustrates the interfaces of the latter, which is based on the basic services provided by the SWKM middleware (based on RDFSuite). The ontologies that are currently in use are available through: http://www.casparpreserves.eu/publications/ontologies/swkmontologies. More about the running prototype are available at http://www.ics.forth.gr/~marketak/GapManager/.



```
«interface»
RepInfoGapManager
+   defineModule(ModuleId, String, String[ ]) : void
+   deleteModule(ModuleId) : boolean
+   getmodules(ModuleId[ ]) : Module[]
+   addModuleTypes(ModuleId, String[ ]) : void
+   removeModuleTypes(ModuleId, String[ ]) : void
+   getDependencyTypes(ModuleId, ModuleId) : String[]
+   updateDependency(ModuleId, ModuleId, String[]) : void
+   deleteDependency(ModuleId, ModuleId) : boolean
+   getDirectDependencies(ModuleId, String[ ], String[ ]) : ModuleId[ ]
+   getDirectDependents(ModuleId, String[ ], String[ ]) : ModuleId[ ]
+   getDirectGap(ProfileId[ ], ModuleId[ ], String[ ], String[ ]) : ModuleId[ ]
```

```
«interface»
DCProfileManager
+   defineProfile(ProfileId, String, ModuleId[]) : void
+   deleteProfile(ProfileId) : boolean
+   getAllProfileIds();() : ProfileId[ ]
+   getProfiles(ProfileId[ ]) : Profile[]
+   getModulesOfProfiles(ProfileId[]) : ModuleId[ ]
+    getProfilesOfModules(ModuleId[]) : ProfileId[]
+   addModules(ProfileId, ModuleId[]) : void
+   removeModules(ProfileId, ModuleId[ ]) : void
```

```
«interface»
DescriptiveMetadataSWManager
+   getDescriptiveMetadata()
+   getDescriptiveMetata(object,Ontology)()
```

Figure 14 . Architecture of the prototype over a Semantic Web Repository

We have already described several data sets according to these ontologies, e.g. the contents of the PRONOM registry. PRONOM[2] is an online registry of technical information that provides information about the file formats, software products, and other technical components required to support long-term access to electronic records and other digital objects of cultural, historical, or business value. PRONOM holds information about file formats and the software products that can process (read, write, identify, etc) each format. Information related to the file formats, such as documentation about them, their compression types, character encoding schemes, and intellectual property rights, is also held. Currently around 600 records are listed.

An important remark is that the core dependency model (sketched in Figure 12) can be considered as an abstraction over different information models: an abstraction that is useful for reasoning about intelligibility. However, the information models themselves may have a richer structure. In that case, the core dependency model can be considered as a *read only* view of such models. For instance, suppose that we have metadata expressed as instantiations of CIDOC CRM ontology and further suppose that this instantiation is expressed in RDF. Not every relationship of a particular instance is an intelligibility dependency. For instance, some descriptive metadata (expressed as types over CIDOC CRM) may not be necessary for the intelligibility of an object. Therefore, issues for further research include (a) identifying relationships or paths for relationships over CIDOC CRM whose semantics is that of intelligibility-dependency and (b) investigating whether additional dependency relationships have to be superimposed over knowledge bases that are structured over CIDOC CRM.

5 CONCLUSION

The preservation of intelligibility is an important requirement for digital preservation. In this paper we formalized this notion on the basis of modules and dependencies. Recall that dependencies are ubiquitous, and dependency management is an important requirement that is the subject of research in several (old and newly emerging) areas, from software engineering to ontology engineering. Subsequently, we formalized the notion of community knowledge in the form of profiles and defined intelligibility and intelligibility gaps. The notion of intelligibility gap is very important as it provides specific answers to questions of the form: What should we be recording and delivering to achieve the intelligibility of digital objects by a specific community? Based on these notions, we sketched a number of intelligibility-awareness processes.

Recall that the preservation of the intelligibility of digital objects requires a generalization (or abstraction) also able to capture non software modules (e.g. explicit or implicit domain knowledge). A modern preservation system should be generic, i.e. able to preserve heterogeneous digital objects, which may have a different interpretation of the notion of dependency. The dependency relations should be specializable and configurable (e.g. it should be possible to assign different semantics to them). Focus should be given to finding, recording, and curation of dependencies. For example, the makefile of an application program is not complete for preservation purposes. The preservation system should also describe the environment in which the application program (and the make file) will run. Recall the four worlds of an information system (Subject World, System World, Usage World, Development World) as identified by Mylopoulos (Mylopoulos, et al, 1990). Finally, the provision of notification services for risks of losing information (e.g. obsolescence detection services) is important.

We described a proof-of-concept prototype based on the Semantic Web. The benefits of adopting Semantic Web languages for the problem at hand are that although the core dependency management services need to know only a very small core ontology (defining the abstract notion of module and dependency), we can refine (specialize) the notion of module and dependency. In this way, we can capture application and domain-specific preservation requirements.

Finally we should remark that described approach can be applied also in cases where conventional languages and formats for preservation (such as EAST, DEDSL, XFDU/SAFE) are employed.

[2] http://www.nationalarchives.gov.uk/pronom/

6 ACKNOWLEDGEMENTS

Many thanks to all partners of the CASPAR project. Also thanks to Martin Doerr for the useful discussions on CIDOC CRM and to Yannis Marketakis for implementing the prototype system.

7 REFERENCES

Adi, A., Etzion, O., Gilat, D., & Sharon, G. (2004) *Inference of Reactive Rules from Dependency Models*. Springer.

Belguidoum, M. & Dagnat, F. (2007) Dependency Management in Software Component Deployment. *Electr. Notes Theor. Comput. Sci. 182: 17-32*.

CIDOC Conceptual Reference Model, *International Organization For Standardization, ISO 21127*:2006.

Franch, X. & Maiden, N. (2003) Modeling Component Dependencies to Inform their Selection. *2nd International Conference on COTS-Based Software Systems*. Springer.

Jarrar, M. & Meersman, R. (2002) Formal Ontology Engineering in the DOGMA Approach. *International Conference on Ontologies, Databases and Applications of Semantics (ODBase)*, 1238-1254. Springer.

Lorie, R. (2001) Long term preservation of digital information, Proceedings of the *1st ACM/IEEE-CS joint conference on Digital Libraries*, 346-352.

Mylopoulos, J., Borgida, A., Jarke, M., & Koubarakis, M. (1990) Telos: Representing Knowledge about Information Systems. *ACM Transactions on Information Systems, 8(4).*

OAIS: Open Archival Information System. *International Organization For Standardization, ISO 14721*:2003, Retrieved from the WWW, February 20, 2009:http://public.ccsds.org/publications/archive/650x0b1.pdf (version of 11 June 2007).

Sunagawa, E., Kozaki, K., Kitamura, Y., & Mizoguchi, R. (2003) An Environment for Distributed Ontology Development Based on Dependency Management, Proceedings of the *2nd International Semantic Web Conference (ISWC2003),* 453-468. Springer.

Tzitzikas, Y. (2007) Dependency Management for the Preservation of Digital Information, Proceedings of the *18th International Conference on Database and Expert Systems Applications, DEXA'2007,* Regensburg, Germany.

Tzitzikas, Y. & Flouris, G. (2007) Mind the (Intelligibility) Gap. *11th European Conference on Research and Advanced Technology for Digital Libraries, ECDL'2007*, Budapest, Hungary.

Vieira, M., Dias, M, & Richardson, D. (2001) Describing Dependencies in Component Access Points. Proceedings of *the 23rd International Conference on Software Engineering (ICSE'01),* Toronto, Canada, 115--118.

M. Vieira, M. & Richardson, D. (2002) Analyzing Dependencies in Large Component-Based Systems. *ASE 2002,* issn = 1527-1366, IEEE Computer Society.

Walter, M., Trinitis, C., & Karl, W. (2001) OpenSESAME: an intuitive dependability modeling environment supporting inter-component dependencies. Proceedings of the *Pacific Rim International Symposium on Dependable Computing,* 76—83.

(Article history: Received 10 December 2008, Accepted 20 February 2009, Available online 30 May 2009)