



Project no. 033572

CASPAR

Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval

Instrument: Information Society Technologies

Thematic Priority: 2.5.10 Access to and preservation of cultural and scientific resources

D2102: Prototype of registry-related KM services



Document identifier:	CASPAR-D2102-RP-0101-1_0
Submission Date:	14-05-2008
Due Date:	31-03-2008
Work package:	WP2100
Partners:	All Partners
WP Lead Partner:	Task (2103) Leader: FORTH-ICS
Document status	FINAL



Abstract:

This document describes the **Prototype of Registry-related KM (Knowledge Management) Services**. CASPAR aims at preserving not only the bits of digital objects but also the information and knowledge that is encoded in these objects. However it is hard to define explicitly what information or what knowledge is. It is therefore very difficult for someone to claim that a particular approach, methodology or technique can indeed preserve information and knowledge. To tackle this issue and for preserving the meaning of digital objects, this document presents a formalization of the notion of *intelligibility* in a OAIS-compliant manner (it actually extends OAIS) – a result of the work of the first phase of this project. The **prototype of registry-related KM services** is founded on this formalization, while its implementation is based on Semantic Web technologies. The document provides some specific and concrete examples of the formalization, gives some guidelines and discusses the functionality of the corresponding software component (along with some testing/validation scenarios).





Delivery Type Software Report

Author(s) Yannis Tzitzikas, Vassilis Christophides, Dimitris Kotzinos, Yannis Marketakis, Grigoris Antoniou
Contributors: CASPAR Consortium.

Approval

Summary

Keyword List

Availability PUBLIC/CONFIDENTIAL(with footnote)/PRIVATE

Document Status Sheet

Issue	Date	Comment	Author
0.1	April 1, 2008	First draft based on of the internal deliverable D2101B.	Yannis Tzitzikas, Dimitris Kotzinos, Vassilis Christophides, Yannis Marketakis, Dimitris Andreou, Grigoris Antoniou (FORTH-ICS)
0.2	April 23, 2008	Various changes plus a new subsection describing the relationship between the Core ontology of GapManager with CIDOC CRM.	
1.0	May 15, 2008	Final deliverable	





Project information

Project acronym:	CASPAR
Project full title:	Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval
Proposal/Contract no.:	IST-2006-033572

Project Officer: Carlos Oliveira

Address:	INFSO-E3 Information Society and Media Directorate General Content - Learning and Cultural Heritage Postal mail: Bâtiment Jean Monnet (EUFO 1167) Rue Alcide De Gasperi / L-2920 Luxembourg Office address: EUROFORUM Building - EUFO 1167 10, rue Robert Stumper / L-2557 Gasperich / Luxembourg
Phone:	+352 4301 33052
Fax:	+352 4301 33190
Mobile:	
E-mail:	Carlos.Oliveira@ec.europa.eu

Project Co-ordinator: David Giaretta

Address:	STFC (formerly CCLRC), Rutherford Appleton Laboratory Chilton, Didcot, Oxon OX11 0QX, UK
Phone:	+44 1235 446235
Fax:	+44 1235 446362
Mobile:	+44 (0) 7770326304
E-mail:	d.l.giaretta@rl.ac.uk





CONTENT

1. INTRODUCTION.....	6
1.1 PURPOSE OF THIS DOCUMENT.....	6
1.2 HOW TO READ THIS DOCUMENT.....	6
1.3 APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS.....	6
1.4 GLOSSARY.....	7
2 TOWARDS FORMALIZING INTELLIGIBILITY.....	11
2.1 MOTIVATION.....	11
2.2 FORMALIZING INTELLIGIBILITY USING MODULES AND DEPENDENCIES.....	11
2.3 EXTENDING THE INFORMATION MODEL OF OAIS.....	17
2.4 SOME DIFFICULTIES AND LIMITATIONS.....	17
2.5 INTELLIGIBILITY-AWARE PROCESSES.....	17
2.6 HOW (OR DIFFERENT WAYS) TO REPRESENT DC KNOWLEDGE.....	19
2.7 SUMMARY.....	21
3 ARCHITECTURE OF SEMANTIC WEB MODELS.....	23
3.1 THE BENEFITS OF ADOPTING SEMANTIC WEB TECHNOLOGIES.....	23
3.2 CORE ONTOLOGY FOR OAIS AND INTELLIGIBILITY-RELATED TASKS.....	23
3.2.1 <i>The Core Ontology for Exchanging Modules, Dependencies and DC Profiles</i>	24
3.2.2 <i>Core Ontology and CIDOC CRM</i>	25
4 GENERAL ARCHITECTURE OF KNOWLEDGE MANAGEMENT SERVICES.....	27
5 REGISTRY-RELATED KM SERVICES (ANALYTIC DESCRIPTION).....	31
5.1 DC PROFILE MANAGER INTERFACE.....	31
5.2 REPIINFO GAPMANAGER INTERFACE.....	32
6 SOFTWARE COMPONENT.....	36
6.1 ITERATIONS.....	36
6.2 MORE DETAILS ON TESTING.....	37
6.3 IMPLEMENTATION DETAILS.....	38
6.4 CASPAR KM INSTALLATION.....	40
6.4.1 <i>Required software components</i>	41
6.4.2 <i>Some Examples</i>	42





1. INTRODUCTION

1.1 PURPOSE OF THIS DOCUMENT

This document describes the **Prototype of Registry-related KM (Knowledge Management) Services**. Recall that CASPAR aims at preserving not only the bits of digital objects but also the information and knowledge that is encoded in these objects. However it is hard to define explicitly what information or what knowledge is. It is therefore very difficult for someone to claim that a particular approach, methodology or technique can indeed preserve information and knowledge. To tackle this issue and for preserving the meaning of digital objects, this document presents a formalization of the notion of *intelligibility* in a OAIS-compliant manner (it actually extends OAIS) – a result of the work of the first phase of this project¹. The prototype of **registry-related KM services** is founded on this formalization, while its implementation is based on Semantic Web technologies. The document provides some specific and concrete examples of that formalization, provides guidelines and methodologies and it discusses the functionality of the software component (along with some testing/validation scenarios).

This document is important for all designers and users of the CASPAR components mainly for those that communicate with the Registry Component.

The latest version of the **software** can be downloaded from

<http://wiki.casparpreserves.eu/bin/view/Main/2103GapManager>

It is structured according to the CASPAR best practices document and it is accompanied by comments, examples and unit tests.

1.2 HOW TO READ THIS DOCUMENT

This document summarizes the main issues results. More information is available in other deliverables as well as in papers published or to be published in scientific conferences and journals (for more see the Section References). More information about the software is available in the web sites that are mentioned in this document.

1.3 APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

Applicable documents

- [S1] Y. Tzitzikas: Dependency Management for the Preservation of Digital Information. 18th International Conference on Database and Expert Systems Applications, DEXA '2007, Regensburg, Germany, September 2007
- [S2] Y. Tzitzikas, G. Flouris: Mind the (Intelligibility) Gap. 11th European Conference on Research and Advanced Technology for Digital Libraries, ECDL '2007, Budapest, Hungary, September 2007

¹ It is worth mentioning that this model has already been published in refereed international conferences and has received encouraging comments





- [S3] Y. Tzitzikas, On Preserving the Intelligibility of Digital Objects though Dependency Management, International Conference PV'2007 (Ensuring the Long-Term Preservation and Value Adding to Scientific and Technical Data), Oberpfaffenhofen/Munich, Germany, October 2007

Reference documents

- [E1] OAIS: Open Archival Information System, International Organization For Standardization, ISO 14721:2003, <http://public.ccsds.org/publications/archive/650x0b1.pdf> (version of 11 June 2007)
- [E2] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis: Telos: Representing Knowledge about Information Systems, ACM Transactions on Information Systems, 8(4), 1990.
- [E3] R.A. Lorie: Long term preservation of digital information, Proceedings of the 1st ACM/IEEE-CS joint conference on Digital Libraries, 346--352, 2001.

1.4 GLOSSARY

[Ax]	Applicable Document
[Rx]	Reference Document
CASPAR	Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval
DoW	Description of Work
EC	European Commission
EPM	Executive Project Management
IPC	IP Coordinator
IST	Information Society Technologies
PACP	Partner Administrative Contact Point
PO	Project Officer
PPR	Project Progress Report
PQE	Project Quality Engineer
PTCP	Partner Technical Contact Point
R&D	Research and Development
SQE	Stream Quality Engineer
ST	Stream
TN	Technical Note
WP	Work Package
WPL	Work Package Leaders





Designated Community	An identified group of potential Consumers who should be able to understand a particular set of information. The Designated Community may be composed of multiple user communities. (OAIS definition)
Archival Information Package (AIP)	An Information Package, consisting of the Content Information and the associated Preservation Description Information (PDI), which is preserved within an OAIS. (OAIS definition)
Content Information	The set of information that is the original target of preservation. It is an Information Object comprised of its Content Data Object and its Representation Information. An example of Content Information could be a single table of numbers representing, and understandable as, temperatures, but excluding the documentation that would explain its history and origin, how it relates to other observations, etc. (OAIS definition)
Knowledge Base	A set of information, incorporated by a person or system, that allows that person or system to understand received information. (OAIS definition)
Representation Information	The information that maps a Data Object into more meaningful concepts. An example is the ASCII definition that describes how a sequence of bits (i.e., a Data Object) is mapped into a symbol. (OAIS definition)
Controlled Vocabulary	A controlled vocabulary is a list of terms that have been enumerated explicitly and is controlled by and is available from a controlled vocabulary registration authority. Usually all terms in a controlled vocabulary should have an unambiguous, non-redundant definition.
Glossary	A glossary is a list of terms, usually with textual definitions. The terms may be from a specific subject field or those used in a particular work. The terms are defined within that specific environment and rarely have variant meanings provided. Examples include the EPA Terms of the Environment.
Dictionary	Dictionaries are alphabetical lists of terms and their definitions that provide variant senses for each term, where applicable. They are more general in scope than a glossary. They may also provide information about the origin of the term, variants (both by spelling and morphology), and multiple meanings across disciplines. While a dictionary may also provide synonyms and through the definitions, related terms, there is no explicit hierarchical structure or attempt to group terms by concept.





Gazetteers	<p>A gazetteer is a dictionary of place names. Traditional gazetteers have been published as books or they appear as indexes to atlases. Each entry may also be identified by feature type, such as river, city, or school. Geospatially referenced gazetteers provide coordinates for locating the place on the earth's surface. An example is the Geographic Names Information Service <http://www-nmd.usgs.gov/www/gnis/>. Note that the term gazetteer has several other meanings including an announcement publication such as a patent or legal gazetteer. These gazetteers are often organized using classification schemes or subject categories.</p>
Taxonomy	<p>A taxonomy is a collection of controlled vocabulary terms organized into a hierarchical structure. Each term in a taxonomy is in one or more parent-child relationships to other terms in the taxonomy. There may be different types of parent-child relationships in a taxonomy (e.g., whole-part, genus-species, type-instance), but good practice limits all parent-child relationships to a single parent to be of the same type. Some taxonomies allow poly-hierarchy, which means that a term can have multiple parents.</p>
Thesaurus	<p>A thesaurus is a networked collection of controlled vocabulary terms. This means that a thesaurus uses associative relationships in addition to parent-child relationships. The expressiveness of the associative relationships in a thesaurus vary and can be as simple as "related to term" as in term A is related to term B.</p> <p>Relationships are generally represented by the notation BT (broader term), NT (narrower term), SY (synonym), and RT (associative or related). Associative Relationships may be more granular in some schemes, e.g. the Unified Medical Language System (UMLS) from the National Library of Medicine has defined over 40 relationships, many of which are associative in nature. Preferred terms for indexing and retrieval are identified. Entry terms (or non-preferred terms) point to the preferred terms that are to be used for each concept. There are standards for the development of monolingual thesauri (NISO, 1998; ISO, 1986) and multi-lingual thesauri (ISO, 1985). However, in these standards the definition of a thesaurus is fairly narrow. Standard relationships are assumed, as well as the identification of preferred terms, and there are specific rules for the creation of the relationships between terms. It should be noted that the definition of a thesaurus in these standards is often at variance with schemes that are actually called thesauri. There are many thesauri that do not follow all the rules of the standard, but are still generally thought of as thesauri. Another type of "thesaurus" represents only equivalence (synonymy), such as the Roget's Thesaurus (with the addition of classification categories).</p> <p>Many thesauri are very large (more than 50,000 terms) and were developed for a specific discipline, or to support a specific product or family of products. Examples include the Food and Agricultural Organization's Aquatic Sciences and Fisheries Thesaurus and the NASA Thesaurus for aeronautics and aerospace-related topics.</p> <p>Other examples: AAT (for arts and architecture) and TGN (for geographic names).</p>





Subject Headings

This scheme provides a set of controlled terms to represent the subjects of items in a collection. Subject heading lists can be extensive, covering a broad range of subjects. However, the subject heading list's structure is generally very shallow, with a limited hierarchical structure. In use, subject headings tend to be pre-coordinated, with rules for how subject headings can be joined to provide more specific concepts. Examples include the Medical Subject Headings (MeSH) and the Library of Congress Subject Headings (LCSH).

Semantic Networks

With the advent of natural language processing, there have been significant developments in the area of semantic networks. Concepts and terms are structured not as hierarchies but as a network. Concepts are thought of as nodes with various relationships branching out from them. The relationships generally go beyond the standard BT, NT and RT (of thesauri). They may include specific whole-part relationships, cause-effect, parent-child, etc. One of the most noted semantic network is Princeton's WordNet, which is now used in a variety of search engines.

Ontology

People use the word **ontology** to mean different things, e.g. glossaries & data dictionaries, thesauri & taxonomies, schemas & data models, and formal ontologies & inference. A formal ontology is a controlled vocabulary expressed in an ontology representation language. This language has a grammar for using vocabulary terms to express something meaningful within a specified domain of interest. The grammar contains formal constraints (e.g., specifies what it means to be a well-formed statement, assertion, query, etc.) on how terms in the ontology's controlled vocabulary can be used together.

People make commitments to use a specific controlled vocabulary or ontology for a domain of interest. Enforcement of an ontology's grammar may be rigorous or lax. Frequently, the grammar for a "light-weight" ontology is not completely specified, i.e., it has implicit rules that are not explicitly documented.





2 TOWARDS FORMALIZING INTELLIGIBILITY

2.1 MOTIVATION

According to the Information Model of the OAIS Reference Model, metadata are distinguished to 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. In brief, the RI of a digital object should comprise information about the Structure, the Semantics and the needed Algorithms for interpreting and managing a digital object. Figure 2-1 shows the corresponding class diagram.

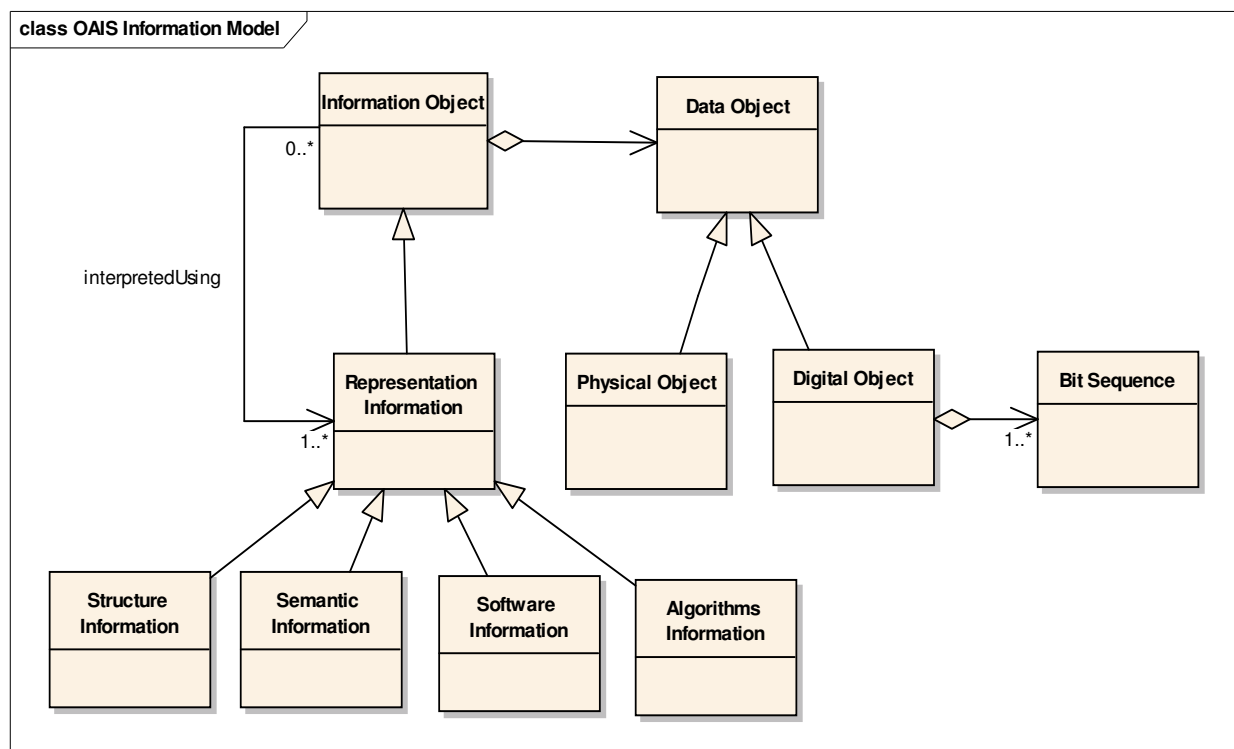


Figure 2-1 The Information Model of OAIS

However for applying OAIS in practice, a number of important questions are raised:

How much representation information should we capture and record?

How this depends on the knowledge of the DC?

How we could model DC knowledge?

What automation could we offer?

2.2 FORMALIZING INTELLIGIBILITY USING MODULES AND DEPENDENCIES

In order to abstract from the various domain-specific and time-varying details, [S1] (i.e. [Tzitzikas, DEXA'2007]) proposed a model for formalizing intelligibility based on the notion of Module and Dependency. A module could be a piece of software/hardware, a knowledge model expressed explicitly and formally (e.g. an Ontology), 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/managed without the existence of t' . So we model the RI requirements of the OAIS information model as dependencies between modules. Some examples are illustrated in Figure 2-2. For instance, the intelligibility of a digital object README.txt depends on the availability of text editors and knowledge of the English language. The rest examples come from the various testbeds of the CASPAR project (e.g. FITS files are used by astronomers). Dependencies also exist between formally expressed knowledge. For instance, the left part of Figure 2-3 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.

An important question that is now raised is how we could model community knowledge according to the above framework. The knowledge of an actor or community u can be characterized by a *profile* T_u that contains those modules that are assumed to be available/known to u (i.e. $T_u \subseteq T$). For example, if u is an artificial agent, then T_u may include the software/hardware modules available to it. If u is a human, then T_u may include modules that correspond to implicit or tacit knowledge. Note that if preservation is done for a particular Designated Community (DC), 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.

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. $Nr(t) = \{t' \mid t > t'\}$. We shall use $>^+$ to denote the transitive closure of $>$, and $>^*$ to denote the reflexive and transitive closure of $>$. Analogously, we can define $Nr^+(t) = \{t' \mid t >^+ t\}$ and $Nr^*(t) = \{t' \mid t >^* t'\}$.

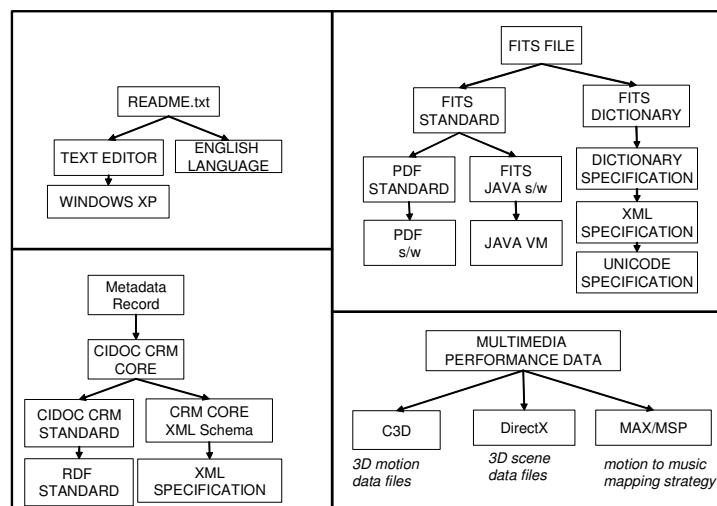


Figure 2-2 Examples of Modules and Dependencies

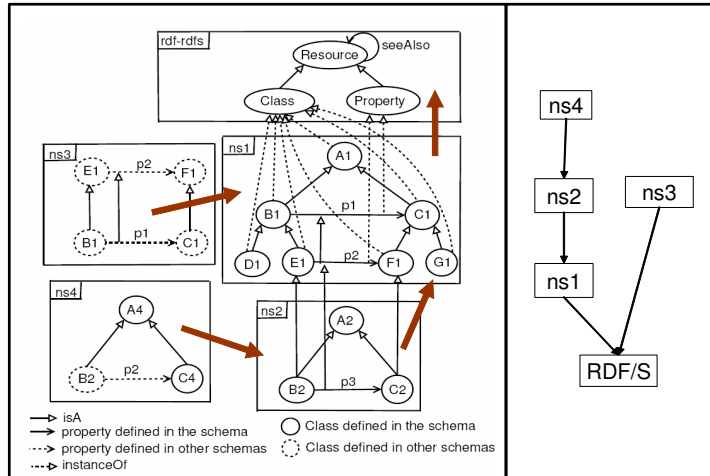


Figure 2-3 Dependencies between RDF Schemas

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 $C(S) = \cup \{ C(t) \mid t \in S \}$. As T_u is a set of modules, we can define its closure as $C(T_u)$.

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 2-4 shows a dependency graph, and the profile T_u of an actor u .

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

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 $Gap(t, u) = C^+(t) - C(T_u)$ 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 T_u . 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: $C^+(t) \subseteq C(T_u) \Leftrightarrow \max(C^+(t)) \subseteq C(T_u)$. In our example $\max(C^+(ty)) = t3 \in C(T_u)$, while $\max(C^+(tx)) = t1 \notin C(T_u)$.

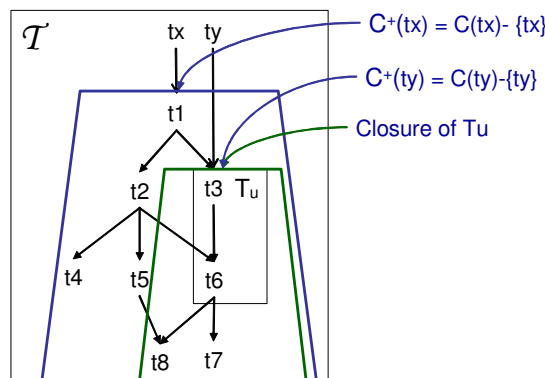


Figure 2-4 Dependency Graph

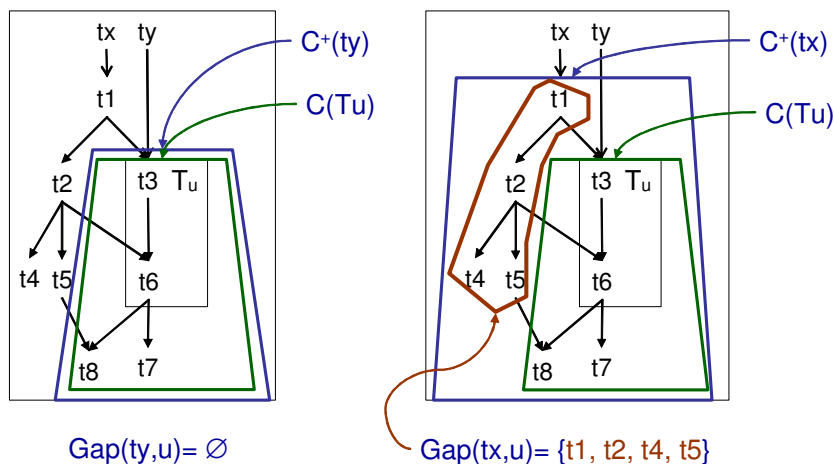


Figure 2-5 Example of Intelligibility Gaps

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 T_u then we need to get and store only $\text{Gap}(t,u)$ plus an id that denotes T_u . Analogously, if we want to deliver an object t to an actor with profile T_u , then the only extra modules that we should deliver to him in order to return him something intelligible, is the set $\text{Gap}(t,u)$.

This means that we don't necessarily have to express explicitly the entire community knowledge. We just have to decide what it is assumed to be the DC knowledge and may introduce modules (which could be symbolic values) that denote this knowledge. This approach enforces us to clarify and express what it is assumed to be known (by the designated community) and what is not.

For example if we have to preserve FITS files for astronomers and we make the assumption that this community knows that the FITS standard and the FITS dictionary is, then we do not have to record any extra RI. If on the other hand we want to preserve these files for ordinary users then we would have to record the dependency graph of Figure 2-2.

As another example consider the example of Figure 2-3. Suppose that $o=ns4$ and that $T_u=\{RDF/S\}$. In this case we have $\text{Gap}(o,u)=\{ns2,ns1\}$. Suppose that T_u is assigned an identifier denoted by $\text{id}(T_u)$, and suppose that $\text{id}(T_u)=\text{profile3}$. In this case what we have to store is $\text{metadata}(o)=\{ns2,ns1,\text{profile3}\}$.

Recall that according to OAIS an AIP (Archival Information Package) is actually a format which consist of the Data Object the required RepInfo plus PDI (Preservation Description Information). 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 allows defining the "right" AIPs. Specifically we can define AIPs that 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 2-6 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.



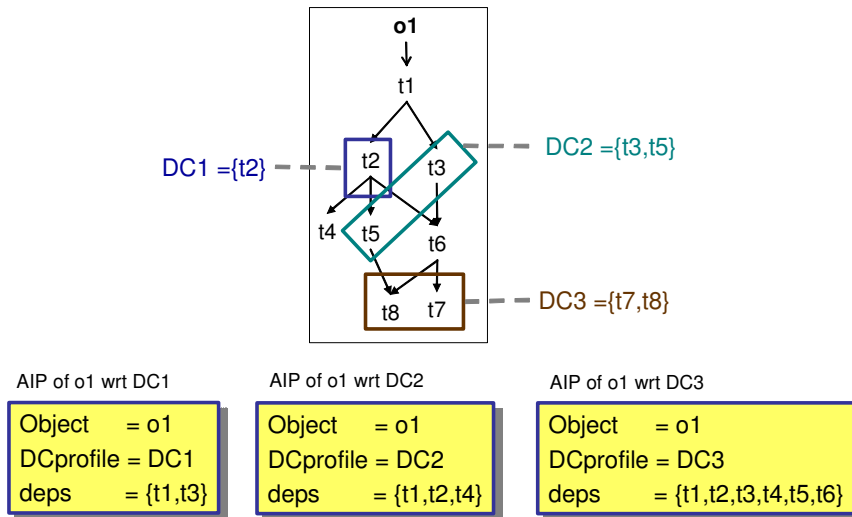


Figure 2-6 Exploiting DC profiles for defining the right AIPs

Of course the less 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 $Tu = \emptyset$, then the problem is just impossible.

But 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 2-7. 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 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 $Gap(b, A) = \{t2, t5\}$.

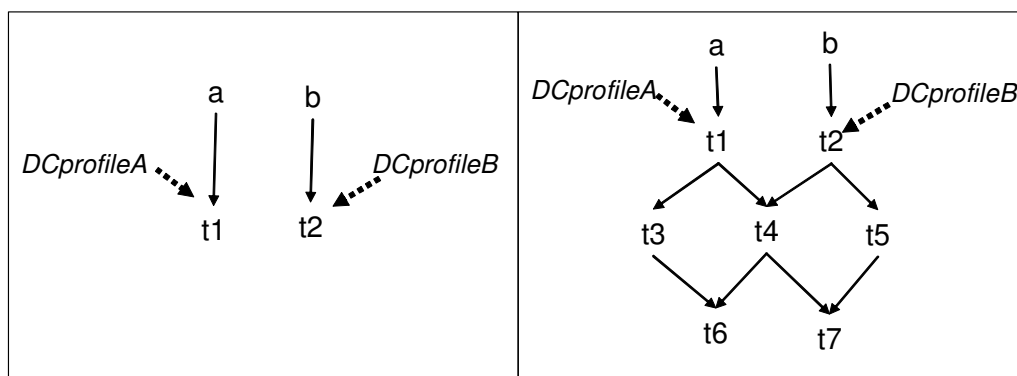


Figure 2-7 Dependencies and Cross-Community Intelligibility

To summarize, the above formalism allowed 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 can he have?

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 2-8). The packaging process is responsibility of the Packaging Component. The Knowledge Manager could aid in providing DIPs according to DC Profiles different that those that have been used for archiving the packages.

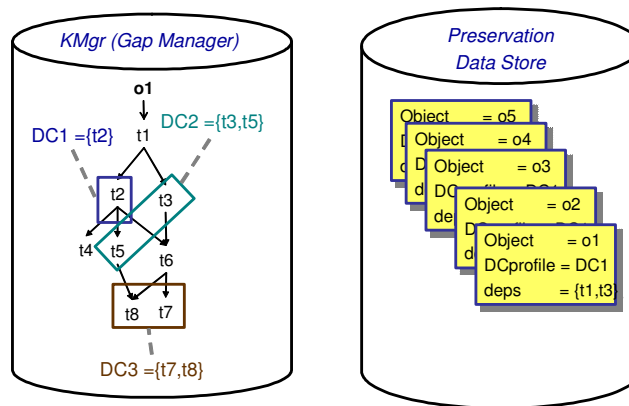


Figure 2-8 Knowledge Manager and Preservation Data Stores

If DC profiles evolve then it would be possible to reconstruct the AIPs according to the latest DC profiles. Such an example is illustrated in Figure 2-9.

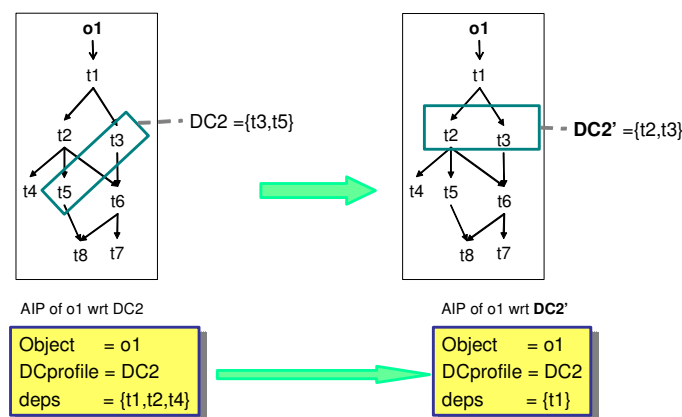


Figure 2-9 Revising AIPs after DC profile changes



2.3 EXTENDING THE INFORMATION MODEL OF OAIS

The above formalization drives to an extension of the Information model of OAIS. In brief the notion of DC Profile should be explicitly represented. A diagram that specifies the extension is shown in Figure 2-10.

Apart from having the additional class Profile and its association with the class Module, this schema suggests having an extensible specialization hierarchy under the class Module. This modelling approach is more flexible than having aggregation hierarchies (as in the original OAIS information model). With aggregation hierarchies one would have to decide and fix at the beginning what kinds of RI there exist. With the proposed approach the kinds of RI may evolve over time. Moreover multiple classification is suggested for the data layer. This means that we may have a module that belongs to more than one subclasses of the class Module. This very useful because in many cases the distinction between structure, algorithms, semantics, etc, is quite blurred.

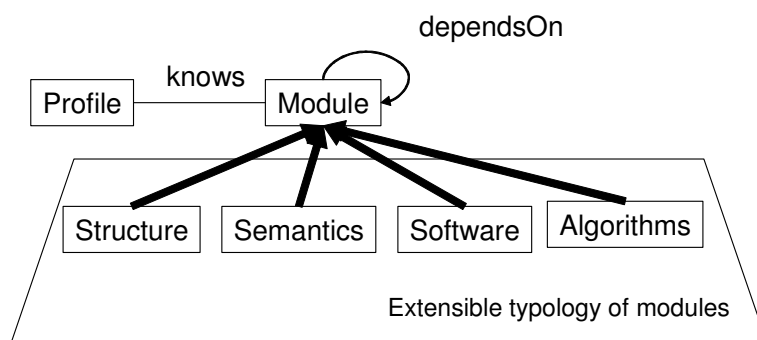


Figure 2-10 Extended OAIS Information Model

The representation of this information model using Semantic Web languages will be discussed in a next section (Figure 3-1).

2.4 SOME DIFFICULTIES AND LIMITATIONS

Let us for example consider the case of Web pages. Consider a digital file named `a.html`. The extension of the filename gives us a hint about the type of the digital object, so we may write `type(a.html)=HTML` and as `a.html > HTML`, we may generalize and consider that for every object `o` it holds `o > type(o)`, if `type(o)` is known. However, an html page is a text that may contain pointers to other types of data (images, sounds, etc). In order to obtain this content, we need a HTML parser. So we could say to compute the dependencies of `a.html` we need to have an HTML parser².

So in practice we may be unable to compute $C(t)$. We may be able to compute only a part of $Nr(t)$.

2.5 INTELLIGIBILITY-AWARE PROCESSES

Figure 2-11 illustrates the functional model of OAIS. The analysis presented in the previous section suggests that a preservation Information System could adopt the notion of profile in order to support intelligibility-aware services. For instance, it could adopt the following policies: (a) the input (e.g. data

² As another example, for a `.java` named file we need to parse the file in order to extract all import statements, while for a `.rdf` named file, we need to parse it in order to extract the namespaces it uses.



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.

OAIS Functional Model

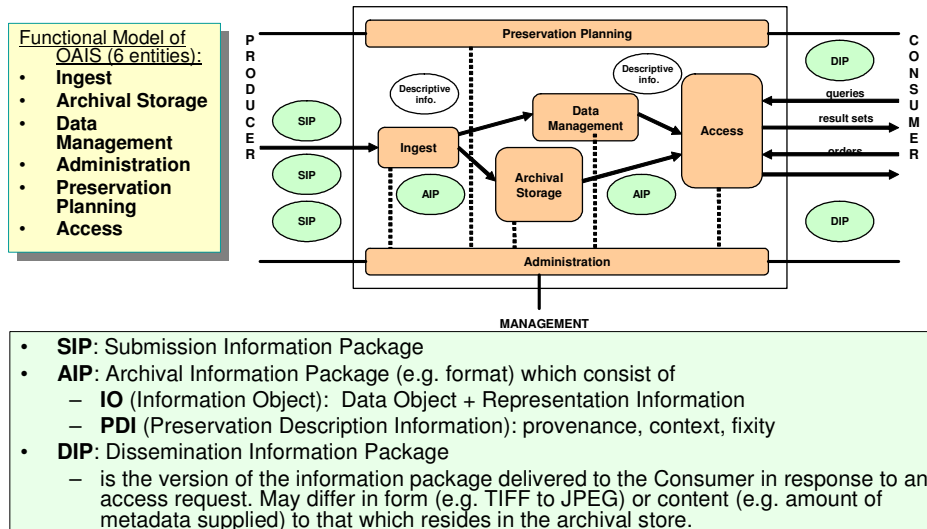


Figure 2-11 The Functional Model of OAIS

Figure 2-12 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 & Flouris, ECDL'2007]. We should remark that these processes correspond to the elements of the functional model of OAIS.

As remarked previously we may be unable to compute the closure of an object, so we may be unable to compute the intelligibility gap. For this reason a progressive/gradual interaction scheme could be adopted (for more details see [Tzitzikas, DEXA'2007]).

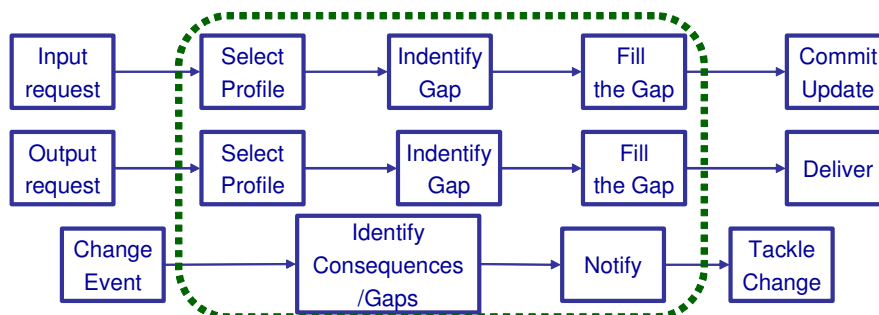


Figure 2-12 Intelligibility-aware Processes



However as they contain intelligibility-related steps, they could be considered **as an extension of the Functional Model of OAIS.**

2.6 HOW (OR DIFFERENT WAYS) TO REPRESENT DC KNOWLEDGE

In general it is very hard to formalize and express explicitly the knowledge of a community. However the previous analysis showed that we don't necessarily have to explicitly represent that knowledge: we may just have to agree on symbols that denote that knowledge. However, the more detailed descriptions we have, the more interoperability and cross-community interpretability we can achieve. Unfortunately in most cases there is a lot of tacit and not explicitly expressed knowledge. This is not always because no one dedicated enough effort to represent explicitly a piece of knowledge. We do not have the machinery and the technology to express everything formally. For example, how to represent the knowledge described in a scientific paper whose subject is second-order-logic?

Due to the above reasons, the formal model of intelligibility that was introduced earlier adopted a very broad definition of what a module can be. A module can be an artifact (e.g. a PDF file describing a data format, or a metadata record instantiating an ontology) or just a symbolic value that denotes something very broad, e.g. the notion of "GreekLanguage". This approach can offer flexibility, i.e. ability to encompass different kinds (in terms of specificity and formalization) of modules. This is a quite pragmatic approach.

Despite the aforementioned problems, in some cases there are artifacts that aim at representing the common conceptualization and the consensual knowledge of a community. There are several forms that such artifacts may have. Some key distinctions are available at the Glossary (at the beginning of this document).

The degree of application and usage should be one important criterion for selecting the appropriate models in case more than one exist, because by definition such models should express consensual knowledge. The definition from scratch is a laborious and not very promising approach. It is better adopting or extending appropriately existing standard models.

Semantic Web languages offer a flexible and standard method to represent several kinds of knowledge. For this reason the next section specifies an Architecture of Semantic Web Models. **However even in that case, these artifacts may depend on other unstructured or tacit knowledge.**

The Tables at the end of this section contain a few examples of different ways for representing (or just denoting) community knowledge.

However it should be remarked that the core model regarding intelligibility (modules and dependencies), i.e. like the one illustrated in Figure 2-10, is advantageous to be expressed in Semantic Web languages and will be detailed later on.

Finally we should say that CASPAR aims at providing insights to the problem of digital information preservation and to investigate the corresponding methodological issues (including the representation of DC knowledge). Its focus is not confined on formalizing the specific knowledge related to the specific datasets of the CASPAR testbeds.

Domain	What could be a module	What could be a Profile
Scientific Knowledge	Module = A scientific paper. Note that the knowledge expressed in the paper may is not possible to be formalized.	Profile = a set of modules. For instance it could be a set of 100 well known and fundamental papers of a discipline. This may mean that the gap of a new paper (i.e. that is needed to be recorded) could be a metadata record (annotation) that relates that





		paper with the aforementioned set of papers.
	Module = a dictionary. It could be a dictionary comprising terms and textual descriptions of the terms. Examples: AAT, CIDOC CRM Ontology, FITS Dictionary. It could be expressed in the form of an RDF file (also containing the appropriate textual descriptions).	Profile = a set of modules. So a set of well known (consensual) dictionaries and ontologies.
	Module = an RDF KB (schemas + instantiations). It could have the form of one file in RDF/XML that contains both schema and data.	Profile = a set of RDF Kbs.
	Module = the knowledge of an individual person. We could consider DavidGiaretta as a module. We could have a dependency of the form: CASPAR > DavidGiaretta	Profile = a set of persons. For example the profile of a community could contain a set of key persons.
Contemporary arts	Module = a MAX/MSP software	Profile = a set of certified tools for running MAX/MSP files
General case of Descriptive Metadata	Consider OAIS provenance. We may describe it by a sequence of derivations and transformations. In that case a module could be the schema that allows expressing these derivations (e.g. CIDOC CRM).	

Some examples of dependencies are shown next

Domain	Example Dependency Relations	
Software Engineering	The spec of an Information System according to MDA should depend on OMG standards (instead of individual technologies). E.g. CASPARSpec > UML 2.0	
Scientific Background	We may say that a publication X depends on all of its citations Y. [Tzitzikas&Flouris, ECDL'2007] > [Tzitzikas, DEXA'2007]	
Multimedia Performance Data	AvisDeTempete.zip > Max/MSPsoftware. However that testbed is currently trying to formalize the logical structure of Max/MSP patches. In that way we will have AvisDeTempeteNew.zip > StandardLogicalStructureOfMax/MSP. This will make it independent of the particular software.	
General	Suppose that an actor A1 wants to preserve the 2D layout of a web page. Another actor A2 may want to preserve also the behaviour of the page.	





	<p>How these differences are reflected to the dependencies?</p> <p>A1: He takes a screen dump and says <code>mypageScreendump.jpg > JPG</code></p> <p>A2: <code>mypageDescriptionOfFlowOfControl > FLOWofControlStandard</code></p> <p>The latter (<code>FLOWofControlStandard</code>) could be an appropriate profile of UML.</p>	
RDF	Note that each RDF file has a header that contain all its dependencies. So we may say that if <code>t</code> is an RDF file then its header lists all ements of <code>C+(t)</code> .	
Java code	The signature of a java class contains its direct superclass (not the indirect). However the specialization hierarchy is not the only aspect of dependency in software code.	

More information about dependencies of INA's works are described in document

- "Representation Information Dependencies: Examples from INA" by Erik Gebers.

Some examples of gaps

Domain	What a Gap could be	
General case	A set of modules, i.e. the result of <code>Gap(o,u)</code>	
A more refined case (typed gaps)	The result of <code>Gap(o,u,X)</code> where <code>X</code> is a subset of the subtypes of Modules. For example we may want to get the missing modules that concern structural or algorithmic information.	
Ontology Diff	<p>Consider the case where <code>Tu</code> contains <code>CIDOCver1</code> and we have a new metadata record according to <code>CIDOCver2</code>. In that case the plain gap manager would return as a gap the module <code>CIDOCver2</code>. However we may employ in this case a more fine grained approach and assume that <code>gap = diff(CIDOCver1-> CIDOCver2)</code> or <code>diff(CIDOCver1, CIDOCver2)</code>.</p> <p>The result of the former is a set of update operations that the actor could apply to his knowledge base in order to reach <code>CIDOCver2</code>.</p> <p>The result of the latter is a set of difference that the person responsible for the archive could investigate in order to judge whether he should migrate his archive <code>CIDOCver2</code>.</p>	

2.7 SUMMARY

The preservation of intelligibility is an important requirement for digital preservation. In this project 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 subject of research in several (old and new emerged) 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 we should be record/deliver to achieve the intelligibility of digital objects by a specific community? Based on these notions we sketched a number of intelligibility-aware processes.





Recall that the preservation of the intelligibility of digital objects requires a generalization (or abstraction) able to capture also 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 different interpretation of the notion of dependency. The dependency relations should be specializable and configurable (e.g. it should be possible to associate different semantics to them). Focus should be given on 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, 1990]³. Finally, the provision of notification services for risks of losing information (e.g. obsolescence detection services) is important.

³ J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis: Telos: Representing Knowledge about Information Systems, ACM Transactions on Information Systems, 8(4), 1990





3 ARCHITECTURE OF SEMANTIC WEB MODELS

3.1 THE BENEFITS OF ADOPTING SEMANTIC WEB TECHNOLOGIES

There are several options for implementing the above framework and the related services. A promising approach is to adopt Semantic Web technologies. The Semantic Web is generally considered to be the next, revolutionary stage of Web technology and in recent years several applications have been developed in numerous fields. The benefits of adopting Semantic Web technologies has been described in the previous deliverables of the CASPAR project, in brief: (1) expressiveness, namely the ability to encapsulate existing metadata schemata and ontologies, as well as new ones (roughly any “classical” conceptual schema can be represented); (2) formal well-foundedness, derived from the classical object-oriented approach and logics; (3) computational amenability (at least for some dialects of Semantic Web languages); (4) world wide scope and impact. The weaknesses of this approach is that (a) there are no industrial strength platforms (so far), (b) the lack of standardized languages/services for KM (especially for supporting Knowledge Evolution), and (c) scalability (most reasoners are main-memory implementations).

Regarding CASPAR, Semantic Web technologies could be adopted in two places:

- 1/ for representing the core OAIS ontology and the information needed for implementing intelligibility-aware processes
- 2/ for representing the knowledge of a community in case this is available or possible.

3.2 CORE ONTOLOGY FOR OAIS AND INTELLIGIBILITY-RELATED TASKS

Figure 3-1 illustrates a possible architecture of SW schemas and data. The upper level comprises 3 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.

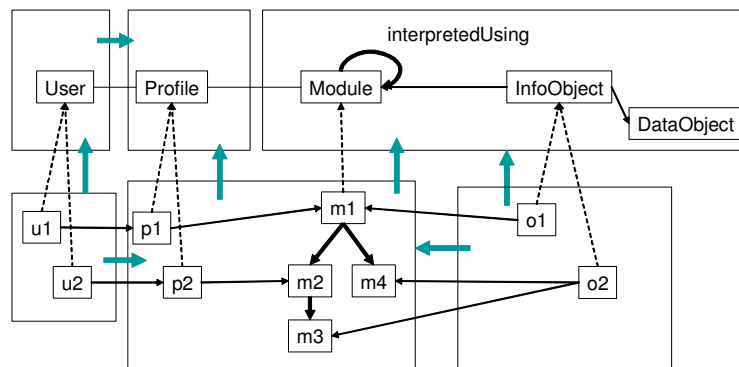


Figure 3-1 Architecture of Semantic Web Models

Between these two layers a number of other schemas can be defined that specialize/refine the elements of the upper schemas. For instance, the notion of module can be specialized (we could have a typology/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*). This is illustrated in Figure 3-2.



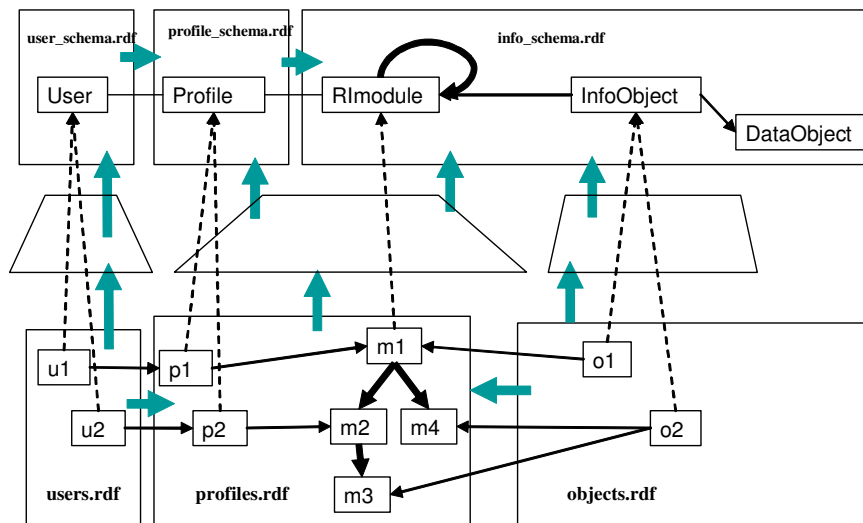
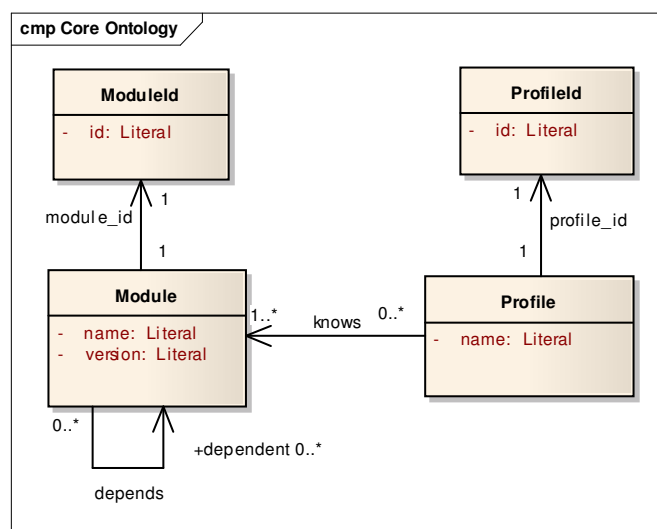


Figure 3-2 Extensible Modeling

One benefit of adopting Semantic Web technologies and an architecture of schemas like 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 will function correctly even if the data level instantiates specializations of the above schemas. This is due to the semantics of specialization.

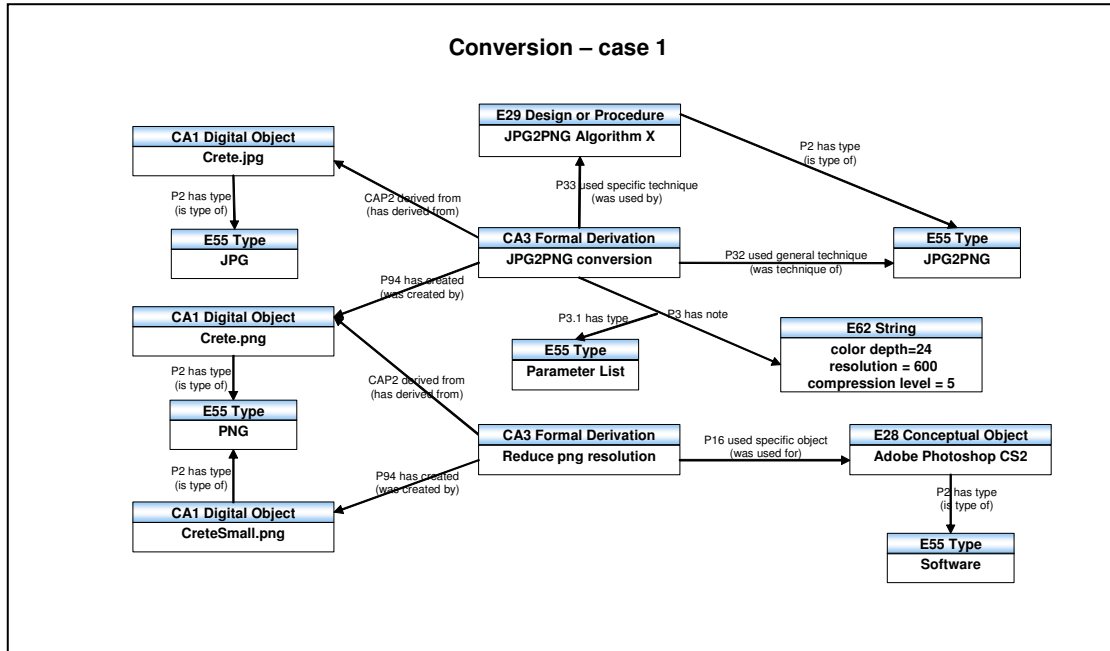
3.2.1 The Core Ontology for Exchanging Modules, Dependencies and DC Profiles

We have already defined (and expressed in RDFS) a Core Ontology for exchanging modules, dependencies and DC profiles. The detailed description of the ontology is given in a separate document. Its description as an UML class diagram is given in the next figure.



We have already used this ontology for describing various data.

The relation *depends* (between Modules) may be mapped to the property **P2 has Type**. An example is given in the next figure that shows 3 images (Create.jpg, Create.png, CreateSmall.png) all defined as instances of the class **C1 Digital Object**. Notice the use of the **P2 has type** relation.





4 GENERAL ARCHITECTURE OF KNOWLEDGE MANAGEMENT SERVICES

A layered KM architecture is more appropriate for this project (and for digital preservation in general). At the lower layer, a general purpose Semantic Web Knowledge Manager (SWKM) could provide a set of core services for managing Semantic Web data. At the upper layer, a CASPAR Knowledge Manager (CKM) will provide high level services based on the OAIS model (and its extensions) aiming at offering an abstraction useful for preservation information systems. CKM can be implemented using services offered by SWKM. The specification of CKM should be generic and less probable to change over time. The lower layer could change in future, or different implementations for it may be available (now or in the future).

This architecture could be used for implementing the intelligibility-related services and the general descriptive metadata services. Below we discuss the latter case.

Consider the case where formally expressed knowledge is available (in the form of ontologies and instantiations). One approach would be to keep these models stored in files (e.g. RDF expressed in XML). However, a more advanced approach is to allow **querying** these models. In such cases a SW repository is proposed. Moreover this approach allows updating the expressed knowledge using declarative languages. However, these approaches are not mutually exclusive.

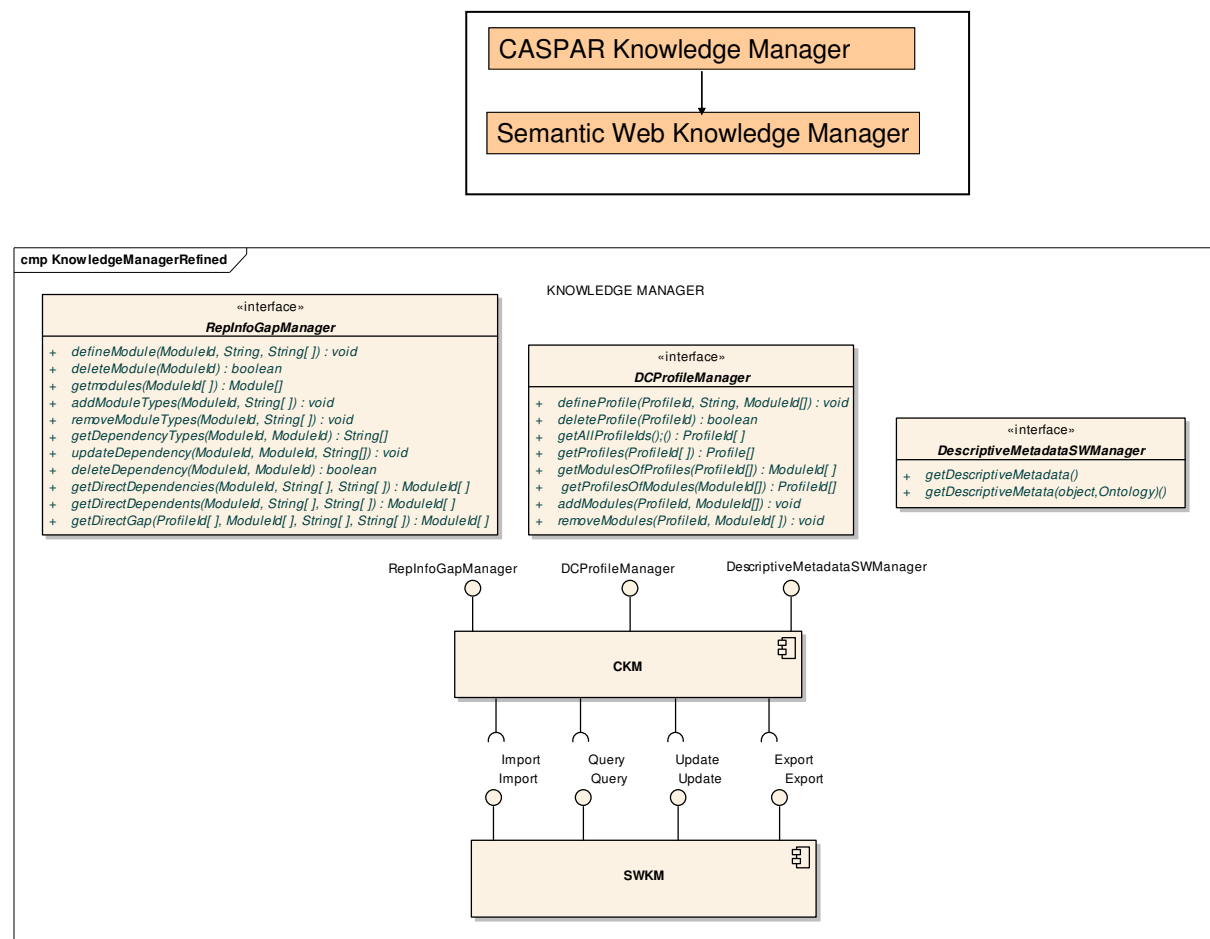


Figure 4-2 Component Model of Knowledge Manager





The description of the components follows

Component	Knowledge Manager
Responsibilities	<p>Capture Higher level Semantics</p> <p>Manage Designated Community Knowledge profile</p> <p>Identify RepInfo Gaps</p> <p>Manage Ontologies and Metadata</p>
Parts	<p>It comprises two layers</p> <p>A) SWKM (Semantic Web Knowledge Manager) which is the lower layer</p> <p>B) CKM (CASPAR Knowledge Manager) which is the upper layer</p> <p>SWKM (Semantic Web Knowledge Manager) will provide a set of core services for managing Semantic Web data.</p> <p>At the upper layer, CKM will provide high level services based on the OAIS model aiming at offering an abstraction useful for preservation information systems. CKM can be implemented using services offered by SWKM.</p>
Provided Interfaces	<p>(A) CKM</p> <p>DCProfileManager</p> <p>RepInfoGapManager</p> <p>DescriptiveMetadataSWManager</p> <p>(B) SWKM</p> <p>Services:</p> <p>Query ()</p> <p>Update ()</p> <p>Import()</p> <p>Export()</p> <p>SWMainMemoryManagement</p> <p>SWMainMemoryModel</p>
Required Interfaces	
Artefacts	<p>The first version of the implementation of the (B)-Services, as a set of web services (based on RDFSuite), has been done.</p> <p>The implementation of a proof-of-concept (A)-level based on the (B)-layer is ongoing.</p>

CKM aims at enabling the intelligibility-aware services that were described in Section 2. It will provide the above specified interfaces:

DCProfileManager

RepInfoGapManager

DescriptiveMetadataSWManager





The following tables sketch the operations of these interfaces. The detailed specification is given in the subsequent sections.

Interface	DCProfileManager
Operations	defineDCProfile updateDCProfile deleteDCProfile getDCProfiles getDCProfileContent
Used by	Data Access Manager and Security

Interface	RepInfoGapManager
Operations	defineDependencies getDirectDependencies getAllDependencies updateDependencies deleteDependencies getRequiredRepInfo(object) getMissingRepInfo(dcprofiles,objects)
Used by	Data Access Manager and Security Registry Preservation Orchestration Manager

Interface	DescriptiveMetadataSWManager
Operations	getDescriptiveMetadata(object) getDescriptiveMetadata(object, ontology)
Used by	Finding Aids

The low level (knowledge repository) basic services should provide scalable persistence services for large volumes of ontologies and ontology-based descriptions. Access and manipulation can be supported by declarative **query** and **update** SW languages. Compared to API-based knowledge application development, the languages supported by the Knowledge Repository aim at satisfying the requirements of CASPAR KM services applications for expressiveness (the ability to express accurately what the query/update initiator wants), generality (the ability to implement easily new query/update functionality) and performances (the ability to respond in a fast way to a query/update request). Following the principle of simplicity and generality only a small number of basic services will be considered. This is for hiding the internal implementation details for making it easier to re-implement these services in the future.





Interface	SWKM Services
Operations	query update import export
Used by	Data Access Manager and Security Finding Aids





5 REGISTRY-RELATED KM SERVICES (ANALYTIC DESCRIPTION)

This section presents a more detailed description of the High Level Services. In particular it presents the interfaces of these services in a way that is independent of any particular implementation.

This specification is a more refined specification of what is presented in the Architecture deliverable.

Beans:

<pre>interface Profile { ProfileId getId(); String getName(); Module[] getKnownModules(); }</pre>
<pre>interface Module { ModuleId getId(); String getName(); String[] getTypes(); }</pre>

5.1 DC PROFILE MANAGER INTERFACE

<pre>// Defines a new profile and associates it with a set of modules (the modules assumed to be known) // Throws ProfileIdAlreadyExistsException if the profile identifier already exists. // Throws ModuleDoesNotExistException if any module identifier specified is non-existent. // If the operation completes successfully, any other defineProfile operation which provides the // same profileId will fail, unless deleteProfile(profileId) is called first. void defineProfile(ProfileId profileId, String profileName, ModuleId[] knownModules) throws ProfileIdAlreadyExistsException, ModuleDoesNotExistException;</pre>
<pre>// Deletes a profile by its identifier, if it exists. Returns whether that profile is existent prior to this call. boolean deleteProfile(ProfileId profileId);</pre>
<pre>// Returns the corresponding profiles of specified profile identifiers. // Throws ProfileDoesNotExistException if a profile identifier is invalid. Profile[] getProfiles(ProfileId[] profileIds) throws ProfileDoesNotExistException;</pre>
<pre>//Returns all existing profile identifiers (which have not been deleted). ProfileId[] getAllProfileIds();</pre>
<pre>// Returns all known module identifiers (in no particular order) of a list of profiles. //Throws ProfileDoesNotExistException if a profile identifier is invalid. ModuleId[] getModulesOfProfiles(ProfileId[] profileIds) throws ProfileDoesNotExistException;</pre>
<pre>// Returns all known profile identifiers (in no particular order) which are currently associated with any module from the ones provided. //Throws ModuleDoesNotExistException if a module identifier is invalid. ProfileId[] getProfilesOfModules(ModuleId[] moduleIds) throws ModuleDoesNotExistException;</pre>
<pre>// Adds a list of modules identifiers in the set of known modules of a profile.</pre>





```
// Throws ProfileDoesNotExistException if the profile identifier is invalid.
// Throws ModuleDoesNotExistException if a module identifier is invalid.
void addModules(ProfileId profile, ModuleId[] modules)
    throws ProfileDoesNotExistException, ModuleDoesNotExistException;

// Removes a list of modules identifiers from the set of known modules of a profile.
// Throws ProfileDoesNotExistException if the profile identifier is invalid.
// Throws ModuleDoesNotExistException if a module identifier is invalid.
void removeModules(ProfileId profile, ModuleId[] modules)
    throws ProfileDoesNotExistException, ModuleDoesNotExistException;
```

5.2 REPINFO GAPMANAGER INTERFACE

```
// Creates a new module with a given identifier, name, and a set of types.
// Throws ModuleAlreadyExistsException if a module already exists with the same identifier.
void defineModule(ModuleId id, String name, String[] moduleTypes)
    throws ModuleAlreadyExistsException;

//Returns the modules which correspond to a list of module identifiers.
//Throws ModuleDoesNotExistException if any module identifier does not exist.
Module[] getModules(ModuleId[] moduleIds)
    throws ModuleDoesNotExistException;

// Deletes a module by its identifier. Returns whether the module identifier actually existed prior to this call.
boolean deleteModule(ModuleId module);

// Adds the specified types to the types of a module.
// Throws ModuleDoesNotExistException if any module identifier does not exist.
void addModuleTypes(ModuleId module, String[] types)
    throws ModuleDoesNotExistException;

// Removes the specified types from the types of a module.
// Throws ModuleDoesNotExistException if any module identifier does not exist.
void removeModuleTypes(ModuleId module, String[] types)
    throws ModuleDoesNotExistException;
```





<p>// Returns the types of the dependency relationship between moduleA and moduleB. // Throws ModuleDoesNotExistException if either module does not exist, or // DependencyDoesNotExistException if there is no direct dependency between these modules.</p> <p>String[] getDependencyTypes(ModuleId moduleA, ModuleId moduleB) throws DependencyDoesNotExistException, ModuleDoesNotExistException;</p>
<p>// Updates or creates a new dependency (if there was none between moduleA and moduleB) // between moduleA and moduleB. //The dependency's types after the successful invocation of this method will be the ones specified. //Throws ModuleDoesNotExistException if either module does not exist.</p> <p>void updateDependency(ModuleId moduleA, ModuleId moduleB, String[] types) throws ModuleDoesNotExistException;</p>
<p>// Deletes a dependency of moduleA to moduleB. Returns whether such a dependency existed // prior to this call.</p> <p>boolean deleteDependency(ModuleId moduleA, ModuleId moduleB);</p>
<p>// Returns a subset of the modules that a specified module directly depends upon. A module X is // included in the returned set if and only if all of the following conditions hold: // (a) a dependency of the form (module depends on X) has been defined and still exists // (b) X contains at least one type included in acceptableModuleTypes, // or acceptableModuleTypes is empty // (c) the types of the dependency relationship (module, X) contain at least one element of acceptableDependencyTypes, // or acceptableDependencyTypes is empty. // Throws ModuleDoesNotExistException if a non-existent module is specified.</p> <p>ModuleId[] getDirectDependencies(ModuleId module, String[] acceptableModuleTypes, String[] acceptableDependencyTypes) throws ModuleDoesNotExistException;</p>
<p>// Returns a subset of the modules which directly depend on the specified module. // A module X is included in the returned set if and only if: // (a) a dependency of the form (X depends on module) has been defined and still exists // (b) X contains at least one type included in acceptableModuleTypes, or acceptableModuleTypes is empty, // (c) the types of the dependency relationship (module, X) contains at least one element of acceptableDependencyTypes, or acceptableDependencyTypes is empty. //Throws ModuleDoesNotExistException if a non-existent module is specified.</p> <p>ModuleId[] getDirectDependents(ModuleId module, String[] moduleTypes, String[] dependencyTypes) throws ModuleDoesNotExistException;</p>
<p>// Returns the direct gap of modules which are needed by a set of profiles to understand a set of modules, // filtered by module and dependency types. This is equivalent to: // - Calculating the union F of getDirectDependencies(m, moduleTypes, dependencyTypes), of // all m in modules. // - Returning F minus getModulesOfProfiles(profileIds) // Throws ProfileDoesNotExistException or ModuleDoesNotExistException if a non-existent profile or // module is specified, respectively.</p> <p>ModuleId[] getDirectGap(ProfileId[] profileIds, ModuleId[] modules, String[] moduleTypes, String[] dependencyTypes) throws ProfileDoesNotExistException, ModuleDoesNotExistException;</p>





Note that currently we under-specify the semantics of [module, dependency] types (currently modelled as strings). We expect that in the future we will understand more thoroughly the level of detail that would be most profitable for defining and manipulating types.

Changes in Dependency Graphs

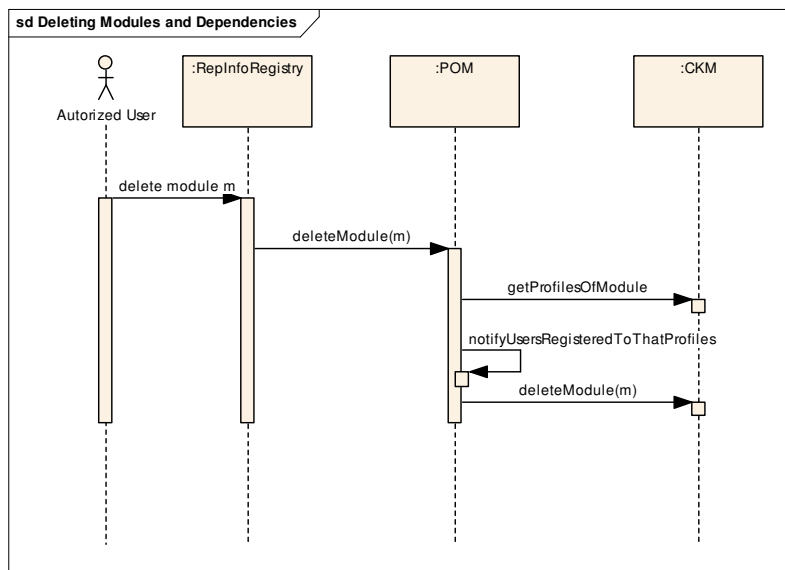
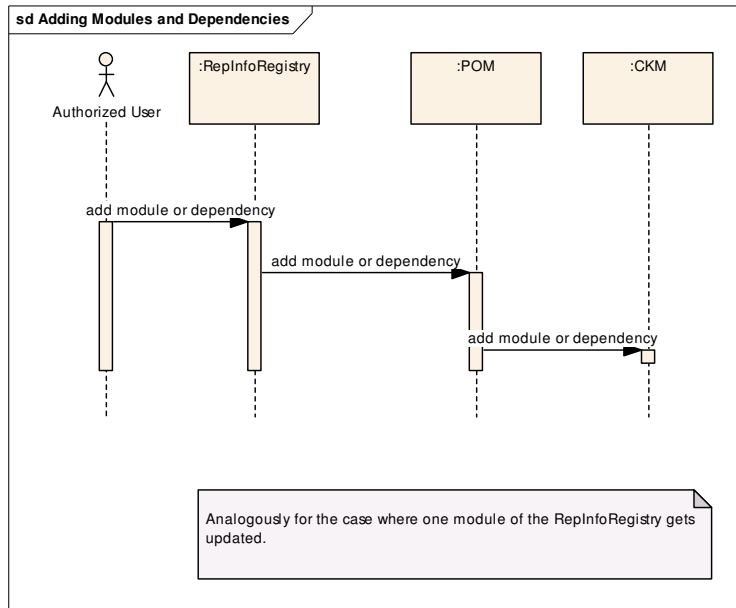
The interfaces **DCProfileManager** and **RepInfoGapManager** are going to be used by the **POM** (Preservation Orchestration Manager). One aspect that has to be analyzed in more detail is the impact of changes on the modules/dependencies/profiles. POM is responsible for the corresponding notification services but what services of CKM are needed has to be analyzed in more detail. Some results on this issue, are available at [S2] (Y. Tzitzikas, G. Flouris: Mind the (Intelligibility) Gap. 11th European Conference on Research and Advanced Technology for Digital Libraries, ECDL'2007, Budapest, Hungary, September 2007). However a more detailed technical specification is required. For instance we have to investigate if the `warnOfImpliedChange` (that POM is requested from CKM) can be based on `getDirectDependents(ModuleId module, String[] moduleTypes, String[] dependencyTypes)` or some additional calls. A preliminary description (on the basis of [S2]) follows:

<i>Simple Operations</i>	Invariants	General Comments
Deletion of module t	All deps to and from t should be deleted? See [S2]	If we assume that the call <code>deleteModule(t)</code> is issued by POM, then POM may also have to call some other methods of RepInfoGapManager so that to identify the affected DC profiles.
Deleting dependencies from the model		Probably there is no side-effect
Adding dependencies to the model		.
<i>Complex Operations</i>		
Upgrade a module with a backwards compatible module It preserves the deps of the previous version	See [S2]	

Another issue is how the **RepInfoGapManager** gets notified when something changes (e.g. additions of modules and dependency relationships). **RepInfoRegistry** and **POM** are the related components. One reasonable approach is the following: Whenever a new module is added at the **RepInfoRegistry** (or any other change), the **RepInfoRegistry** informs POM and then **POM** informs **RepInfoGapManager**. Then the POM could issue the appropriate calls to CKM in order to identify who other actors should be notified.

Indicative interaction diagrams follow.







6 SOFTWARE COMPONENT

6.1 ITERATIONS

The main functionality of each iteration and some testing/validation methods are described in the following table. The table includes not only the Registry-related KM services but also the basic KM services on which they depend.

Iteration	Functionality	Motivation	Testing/Validation Method(s)
I1	First version of the Basic SWKM Services. Options: 1/ Full installation 2/ Installation of only the client.	The high level knowledge management services (e.g. RepInfoGap Manager) will be based on the these. In addition, the rest CASPAR components may exploit these basic services.	I1TC1/ Ability to build a SW repository offering persistence, validation and query and update services. I1TC2/ Ingestion test: one partner from the testbeds (e.g. IRCAM) uses these services in order to feed the repository with some data (e.g. with the CIDOC CRM ontology and descriptions of some indicative objects with respect to that ontology). I1TC3/ The component Access Manager uses these services to offer some content-based access services. E.g. provides some provenance queries assuming the CIDOC CRM ontology.
I2	(2a) Revised version of the Basic SWKM Services including a design and a first implementation of a Main Memory Model. (2b) First version of the Gap Manager	RepInfo Gap Manager is instrumental for providing intelligibility-aware services	I2aTC1/ Ability to support some forms of knowledge evolution (recall the example of Yugoslavia). I2aTC2/ Provision of an API for managing Semantic Web data in main memory. I2bTC2/ Ability to implement the examples described in this deliverable regarding intelligibility gaps. Definition of modules, profiles, dependencies, etc. Demonstrating examples that users with different profiles get different responses.
I3	Second version of Gap Manager	Interaction with other caspar components. GapManager should actually receive requests from other components. It is not expected to call other components.	I3TC1/ The registry or POM feeds the KM Repository (with modules and dependencies). I3TC2/ The component Preservation Orchestration Manager uses the RepInfoGapManager services.
I4	Diff over Knowledge Bases	Support of knowledge evolution is important and it should be demonstrated.	I4TC1/ It takes as input two versions of one ontology (say V1 and V2 of CIDOC CRM). The tool should be able to identify the changes and derive a set of change operations which could be applied on a knowledge repository on CIDOC CRM V1 in order to reach CIDOC CRM V2.

Table 1 Functionality of each iteration





6.2 MORE DETAILS ON TESTING

Each test scenario is checked by writing the corresponding JUnit tests. In this way it is possible to run them periodically and automatically. This is important for ensuring the correctness of the code as the project proceeds and new functionality is added. For each of the testing scenarios of the previous table a more detailed set of test cases are given in the following table. Of course, more test cases will be designed and developed as the project proceeds.

IIT C1	No	Ability to build a SW repository offering persistence, validation and query and update services.	
		Try to import a valid (syntactically and semantically) expression of CIDOC CRM in RDFS to the repository. The import request should succeed and the correct internal structures of the repository are created.	
		Try to import an invalid (syntactically or semantically) expression of CIDOC CRM in RDFS to the repository. The import request should fail and the KM repository should remain intact. This test will ensure that the import requests have transaction semantics.	
		Try to import a namespace B that extends a namespace A that is already imported. The request should succeed. If A is not already imported the request should fail.	
		Try to export from the repository an ontology that has already been imported. A valid output file should be returned. The output file should be possible to be imported to the KM repository.	
		Try to export from the repository an ontology with all its dependencies (i.e. with all other ontologies that it extends). The result should be a set of files that someone should be able to import to a new KM repository.	
		Try all combinations of pre/post conditions of the specification of the basic services (that are described in Section Error! Reference source not found.)	
IITC2		Ingestion test: one partner from the testbeds (e.g. IRCAM) uses these services in order to feed the repository with some data (e.g. with the CIDOC CRM ontology and descriptions of some indicative objects with respect to that ontology).	
		Try to import some valid instantiations of CIDOC CRM expressed in RDF. The import request succeeds and the correct internal structures of the repository are created. A failure should be reported if the file to be imported is syntactically or semantically incorrect.	
		Try to import some valid instantiations of CIDOC CRM expressed in RDF. The import request should succeed and the correct internal structures of the repository should have been created.	
		Use the query language that verify that a class c1 is subclass of a class c2 for the case where c1 and c2 have been defined in different namespaces and c2 has been defined as an extension of c1 in the corresponding RDF file that was imported.	
IITC3		The component Access Manager uses these services to offer some content-based access services. E.g. provides some provenance queries assuming the CIDOC CRM ontology.	
		Some indicative queries assuming CIDOC CRM are formulated. The KM repository may have stored instances according to a specialization of CIDOC CRM (e.g. wrt a schema B that extends the concepts of CIDOC CRM). The result of the query should not be empty.	
I2aTC1		Ability to support some forms of knowledge evolution (recall the example of Yugoslavia).	
		Test the scenario of Yugoslavia. Formulate a number of queries that ensure that the KB has been updated. Ensure transaction semantics.	





I2aTC2		Provision of an API for managing Semantic Web data in main memory.	
		Try exporting a namespace that is already stored in the KM repository. From the returned file it should be possible to create a main memory model (MMM) that an application programmer could use in order to build an application. Specifically the MMM should allow someone to get the classes, the properties, the resources and the property instances that are defined in the exported file.	
I2bTC2		Ability to implement the examples described in this deliverable regarding intelligibility gaps. Definition of modules, profiles, dependencies, etc. Demonstrating examples that users with different profiles get different responses.	
		For each of the methods described in Section 5 a test case will be created on the basis of the pre/post-conditions that are already specified.	
I3TC1		The registry or POM feeds the KM Repository (with modules and dependencies).	
		The repository stores and responds to the requests according to the specification (related: I2bTC2). The messages received from POM should be manageable by RepInfoGapManager.	
		POM is notified by the RepInfoRegistry about a change of a module. POM should be able to call the appropriate methods of CKM in order to identify the affected DC profiles. This will ensure that the design specification is complete.	
I4TC1		It takes as input two versions of one ontology (say V1 and V2 of CIDOC CRM). The tool should be able to identify the changes and derive a set of change operations which could be applied on a knowledge repository on CIDOC CRM V1 in order to reach CIDOC CRM V2.	
		This scenario involves a lot of research. For this reason this scenario has a "WANT" priority (according to Must Should Could Want scheme for setting priorities) <ul style="list-style-type: none"> 1. Take as input two namespaces A and B expressed in RDFS 2. Compute the Diff between two A and B as a sequence of change operation DU 3. We have to test whether the application of DU on a knowledge repository that contains A will result to one namespace that is semantically equivalent to A Several experiments (with various different pairs of A and B namespaces) will be conducted.	

Table 2 Test Cases

Timescale

Iteration	Input	Start Date	End Date
I1	All CASPAR deliverables	M6	M23
I2	D2101B	M18	M27
I3	Revised architecture	M28	M31
I4	Instantiations of CIDOC CRM and its extensions.	M28	M35

Table 3 Schedule of Iterations

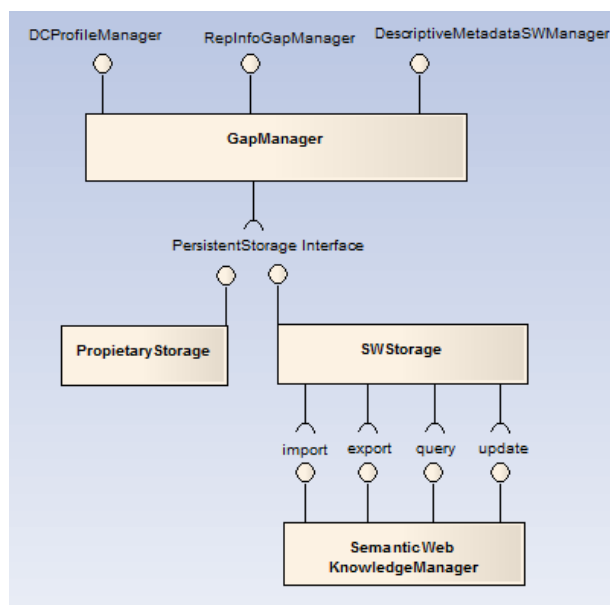
6.3 IMPLEMENTATION DETAILS

Gap Manager





The implementation 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 the Semantic Web Knowledge Middleware (over the basic SWKM services). The design of this module is illustrated below.



The current implementation of Gap Manager supports more than one input and output formats. Details follow:

- Proprietary Format

This is a very simple and human-readable format. Three different files are used (one for modules, one for profiles and one for dependencies). At the modules file each line corresponds to one module. At this line we keep all the information needed using a simple syntax where tabs are used to separate the attribute values. Specifically the first attribute is the module identifier. After that is the name and the version of the module. The rest of the line is used for keeping the types of a module. All the above are separated using tabs ('t'). The dependencies file contains all the dependencies between modules described previously. The syntax here is the same as before. The first two attributes are the identifiers of the involved modules and the rest of the line is used for specifying the dependency types (can be more than one). At profiles file, similarly to modules, each line represents one profile. The profile identifier and name are the first in the line. Subsequently we list all the identifiers that the particular profile knows.

- Pronom Format

Using the xml file exported from Pronom we defined a utility to import data from this file format. The data we can get concern file formats and software. Specifically there are two different xml files (coming from Pronom). We use our utility to import data from these files and get them as modules, since we recognised both file format and software as modules. There are information that we do not currently use (such as external signatures, risks, and internal identifiers for file formats and software) at these files.

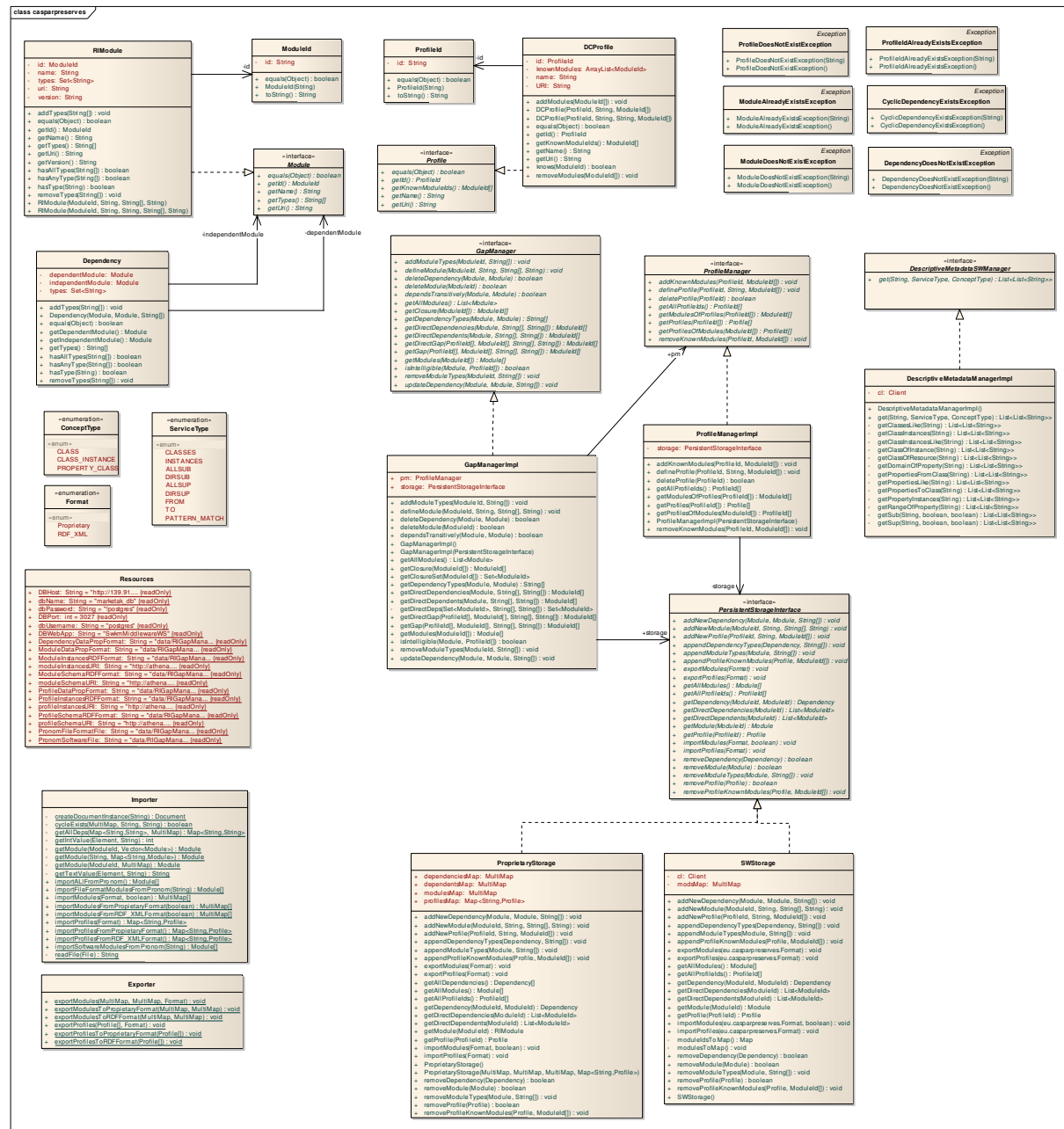




• Ontology for Modules, Dependencies and DC Profiles

We defined an ontology (in RDFS) for describing the modules, the dependencies between them and the DC profiles. This ontology consists of two schema files and other files containing instances. The first file (module_schema.rdfs) contains information about the schema of the modules and their dependencies. The other file (profile_schema.rdfs) describes the schema of the DC profiles. Any other file contains instances according to these schema files. More information can be found at [S1].

The class diagram of the Gap Manager implementation is shown in the next Figure.



6.4 CASPAR KM INSTALLATION





6.4.1 Required software components

The Prototype of the Knowledge Manager is based on Java based web services (SWKM services from now on) running on top of a working Glassfish installation. This implies that a working Java Development Kit installation (at least Java 5.0) should be already present or downloaded and installed from <http://java.com/java/download/index.jsp>.

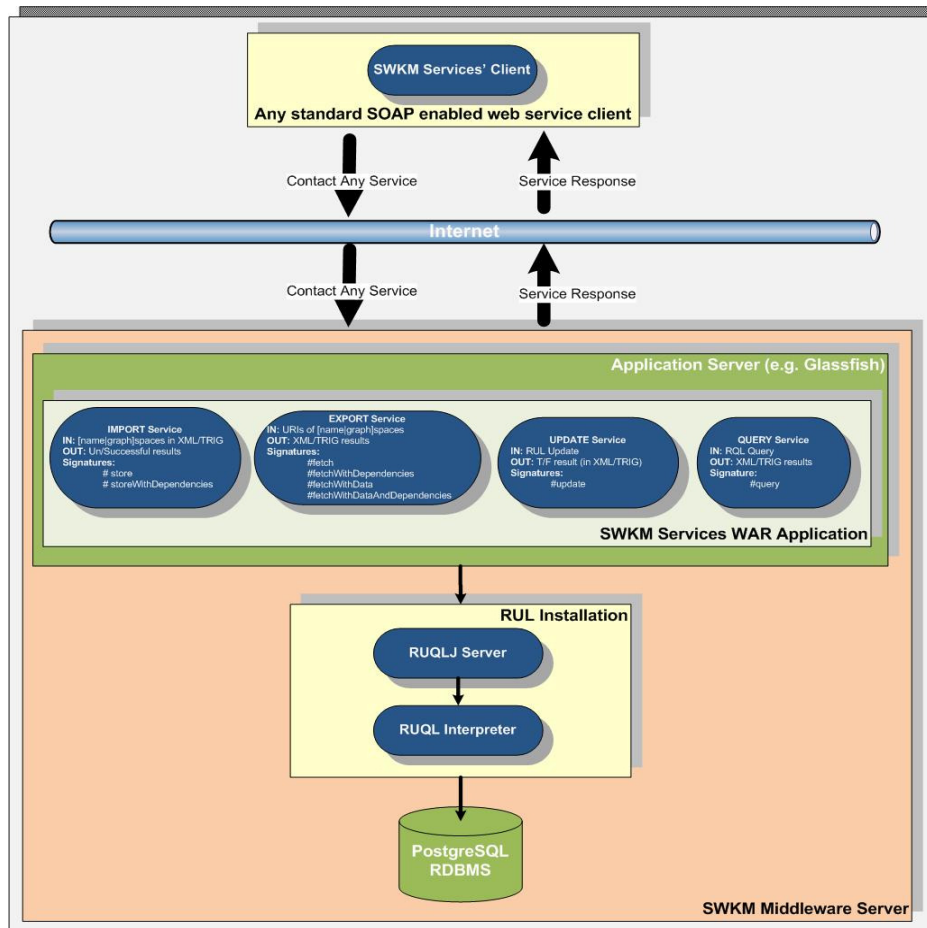


Figure 1. The various software components required by the SWKM services

In detail the required components for SWKM services to run and their various connections and dependencies as well as the available services after deployment are depicted in Figure 1.

The latest releases of software and documentation for the Semantic Web middleware (basic services and client) is continuously available at <http://athena.ics.forth.gr:9090/SWKM/>.



The Gap Manager is based on the Client described previously. The latest version of the software can be downloaded from

<http://wiki.casparpreserves.eu/bin/view/Main/2103GapManager>

It is structured according to the CASPAR best practices document and it is accompanied by comments, examples and unit tests.

6.4.2 Some Examples

The following figures illustrate some parts of the functionality offered by the Registry-related KM Services. Specifically they demonstrate how they could be used in order to derive intelligibility-aware Dissemination Information Packages (DIPs).

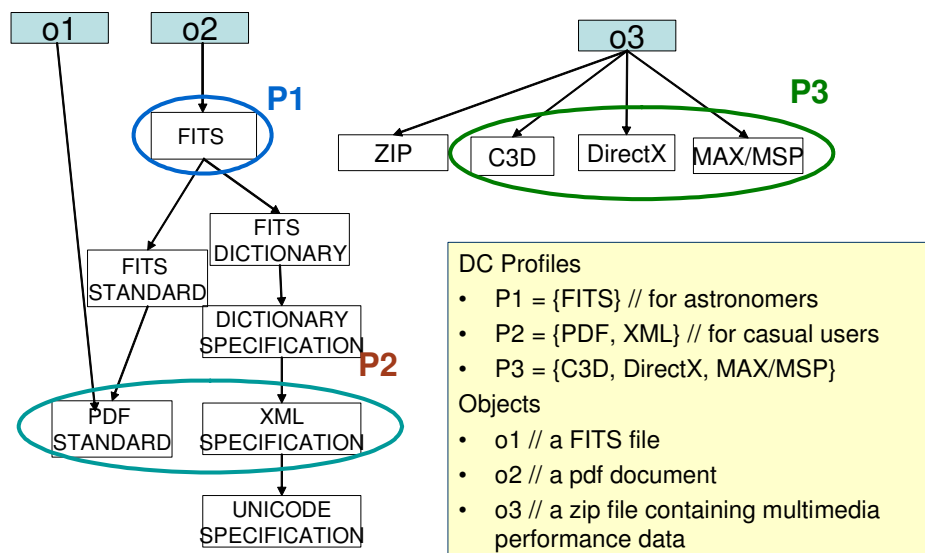


Figure 3 Example of modules, dependencies and profiles

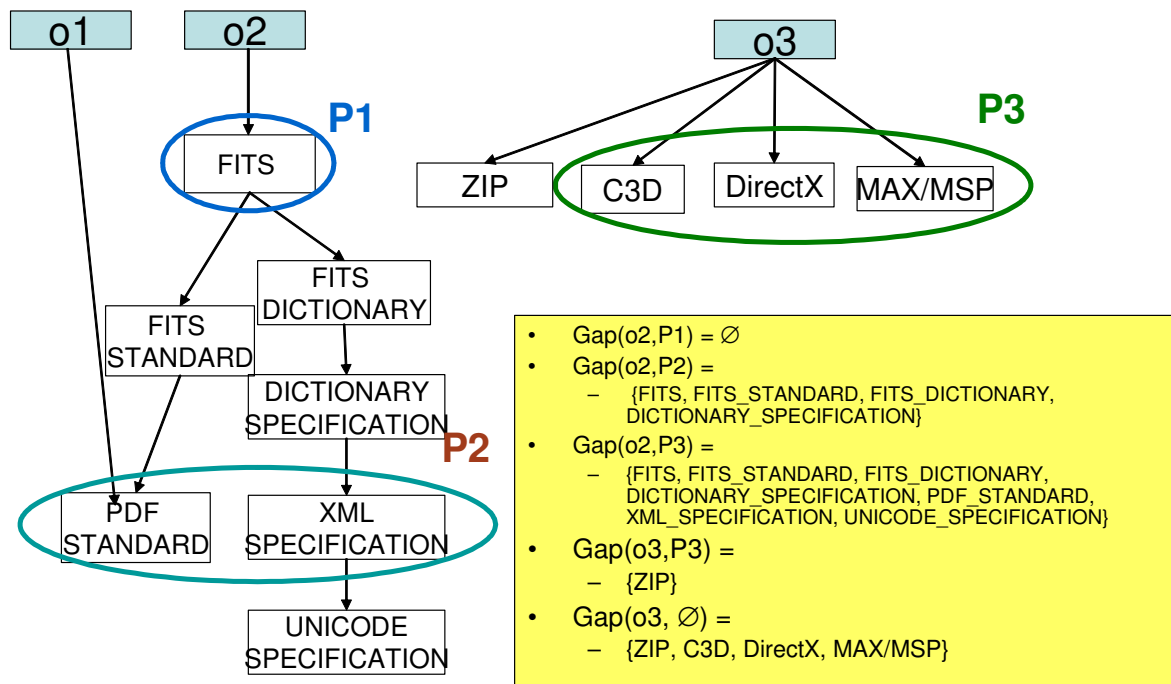


Figure 4 Examples of Intelligibility Gaps

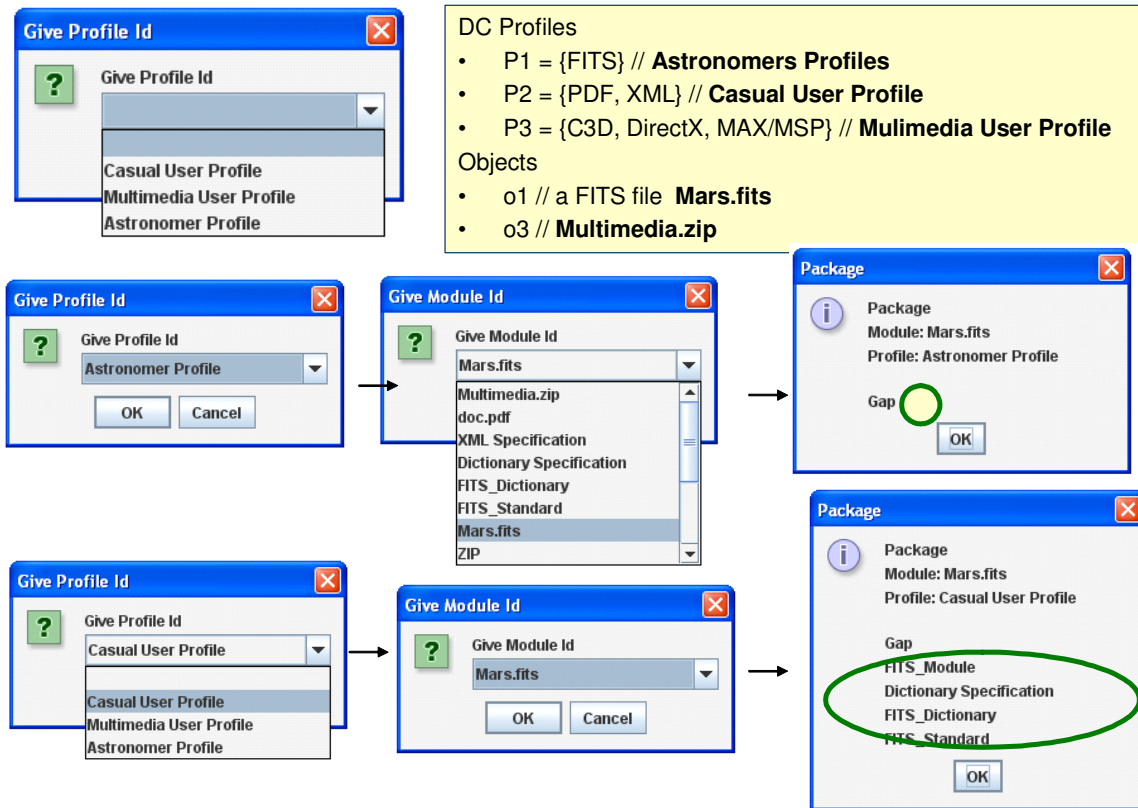


Figure 5 Screenshots of an indicative working prototype

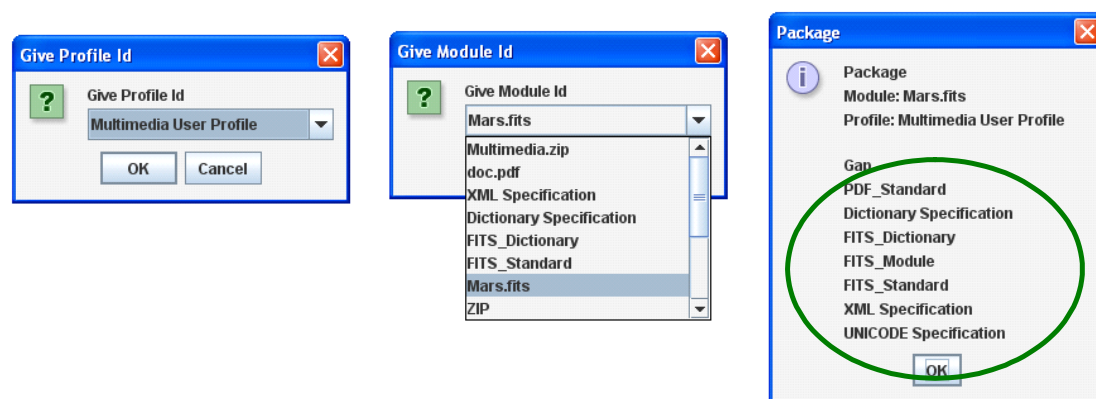


Figure 6 Screenshots of an indicative working prototype