SCIENTIFIC KNOWLEDGE OBJECTS V.1

Fausto Giunchiglia and Ronald ChenuAbente

January 2009

Technical Report # DISI-09-006

Scientific Knowledge Objects v.1

Fausto Giunchiglia[1], Ronald Chenu-Abente[1]

[1] University of Trento, Dipartimento di Ingegneria e Scienza dell'Informazione, 31800 Trento, Italy
{fausto, chenu}@disi.unitn.it

Abstract. This document introduces the SKO and its associated structures as a response to the needs of a collaborative platform for the creation, dissemination and publication of Complex Artifacts and also as an option to the current paper-centered scientific publication practices. The approach presented is based on three Organization levels (Data, Knowledge and Collection) and also three States (Gas, Liquid, Solid) that regulate the properties and operations allowed at each level.

Keywords: Complex Artifacts, Paper Publishing, Data organization, Knowledge, Collection, Liquid, Solid, Gas.

A Knowledge Artifact is an object created as a result of an activity which encodes knowledge, the understanding or awareness gained beyond data. On the other hand, Complex Artifacts are those that are composed from several simpler Artifacts that have been made into a single coherent unit. Examples of these Complex Knowledge Artifacts include scientific papers, books, and journals, among others.

In this document define and detail the Scientific Knowledge Object (abbreviated as SKO) introduced at the Liquidpub proposal [1]. These will become the unit that will be used to represent the Complex Knowledge Artifacts in the context of the Liquidpub project.

As a direct response to the current problems and deficiencies treated in the Publish and Perish paper [2], the following are among the most important properties of the SKO:

- *Complex structure*: because SKOs may contain different types of data (texts, spreadsheets, images, videos, etc) along with its corresponding metadata.
- *Composabilty and Reusability*: a SKO built from lower-level structures which, at the same time, may be used by several other SKOs.
- *Evolvability*: either the SKO as a whole or its different components can change and evolve over time.
- *Facilitate collaborative work*: the previous properties like evolvability and composability greatly facilitate the creation of SKOs as collaborative effort between different actors.

Along with the SKO other two main structures will be introduced; the SKOnodes which represent the data pieces that the SKO brings together; and the SKOsets, which represent collections and aggregations of other structures.

Each of these structures represents a different conceptual and organizational level, as displayed at the next table:

Table 1. The three proposed levels of organization

Level	Name	Unit	Purpose
1	Data	SKOnodes	Organize data
2	Knowledge	SKOs	Joining and Ordering of data and knowledge to to propose new knowledge.
3	Collection	SKOsets	Categorize related data and knowledge

Furthermore we add a different dimension to this three-leveled organization approach by introducing three States; Gas, Liquid and Solid; a metaphor to the States of physical matter itself. Each of the mentioned States can be assigned to each of the mentioned organizational level structure, which fundamentally changes their properties and allowed operations.

This document will start with introduction of the three levels of organization; Data(Chapter1), Knowledge(Chapter2) and Collection(Chapter3); along with other additional structures(Chapter4) that complement the previous three.

The State dimension will be introduced (Chapter5) detailing the purpose and properties introduced by each state, along with a treatment of the change-state operations for the defined structures. Finally, (Chapter6) will deal with the Basic Operations for the structures, always detailing how the current State of the structure affects it.

More information about SKOs and the Liquidpub project can be found at the Liquidpub project homepage [3].

1 The Data Representation Level

Data organization is the most basic level of organization and its final objective is to organize the data into semantically linked nodes. It does so by taking each data piece, adding a metadata structure along with an unique id URL and links to create relations between these nodes. The final result of this process is the creation of a Directed Acyclic Graph (DAG) that includes all the original data.

The chapter will first give the definition of the basic unit of this level, the SKOnode, along with its components and it will also present considerations about the structures that can be built by using these SKOnodes.

1.1 SKOnode Definition

This section will define the SKOnodes, the node structure to be used to organize data at this level. The following graphical representation gives a basic idea of the components of the SKOnode structure:

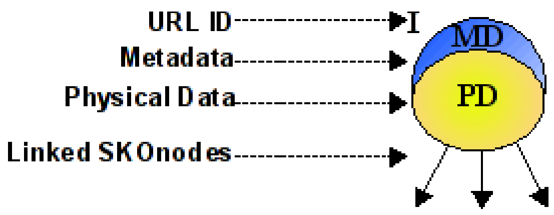

Fig. 1. Graphical representation of a SKOnode

More specifically, a SKOnode N is defined as:

$$N = <URL, MD, PD, \{LinkedNs\}> . \quad (1)$$

Where:

- *URL*: provides a unique identifier to the node.
- *MD*: metadata attributes that are used to contain information related to the node and its descendants. The general structure and function of metadata and attributes will be given later on section 4.1.
  As an additional requirement for the SKOnode's MD:
  - *State*: a *unique* attribute with the Name or Label "State" *must always* be defined. Furthermore, its value *must be* either "Gas", "Liquid" or "Solid". More information about State and its uses will be given later on Chapter 5.
- *PD*: PhysicalData points to the actual Data of the node which for example, could be a html file, a filesystem file, etc. Nodes with empty or null values of PD are not allowed.
- *{LinkedNs}*: a possibly empty (0...n) set of the SKOnodes pointed by the current node.
  The links from {LinkedNs} carry a semantic meaning, as all the nodes inside of the LinkedNs set from N are considered to be *"Part of"* N in such way that P{LinkedNs}N holds.
  An additional requirement is related to the {LinkedNs} component:
  - *Cycle exclusion*: For each node Nn to be added to the set {LinkedNs}, a non-empty directed path that starts and ends on Nn must not exist.

Depending on the decided implementation of the previous structure, all the other components that come after MD can be chosen to be implemented as attributes contained in MD.

Some basic examples of single SKOnodes include text, data sets, stimuli, figures, images and other non-Complex Artifacts.

1.2 SKOnode Semantic Structure

Based on the previous definition, this section will introduce the interactions and structures that are made possible by the joining and interaction of SKOnodes

Starting from a Complex Artifact A with d1, d2, ... , dn representing all its data (for example, text, images, etc.) it is possible to create SKOnodes by adding an URL and a metadata structure to each of A's data units (Nn = <URLn, MDn, dn, 0>). This process could be represented as in the following image:

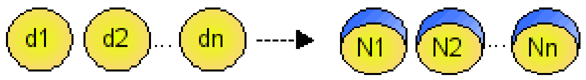

Fig. 2. Conversion of data units into SKOnodes, the light blue portion of each node represents the metadata of the node while the bottom yellow part represents its data.

To join these nodes and contain general information about them a Root node Nr = <URLr, Mdr, dr, {N1, N2, ..., Nn}> can be created. Note how this node's {LinkedNs} is defined in such way that it has “Part of” links to each of the originally defined Nn nodes. With the introduction of a root, the resulting structure can be further refined by reordering the nodes and giving more depth to the structure according to the actual contents of each node.

Though it may seem that by applying the previous procedure a Tree will always be created, the SKOnode definition does not forbid linking to the same node from multiple ones. Semantically this means that each node may be “Part of” of two or even more nodes, allowing the resulting structure become a rooted DAG instead.

The previous leads us to the following lemma:

If A is a Complex Artifact then for all A a rG exists such that rG is a *rooted DAG* of SKOnodes and it includes the same data as A. (**2**)

For example, when transforming a document with a clearly defined Title, Chapters and Paragraphs of text into SKOnode rooted DAG, the result cab be a structure similar to the one in the following example:

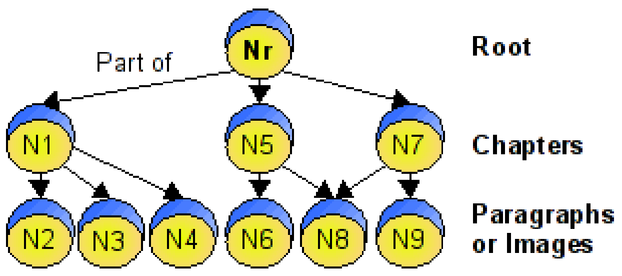

Fig. 3. Example of a SKOnode structure resulting from the conversion of a Complex Artifact

The SKOnode-based semantic structures are particularly interesting because, when writing complex documents, authors that take a top-bottom approach can first define their index(high-level nodes) and work on its refining (creating lower level nodes). On the other hand, authors that take a bottom-up approach would not have problems creating several unrelated nodes of data to later join them with the creation of higher-level nodes.

1.3 The Universal SKOnode Graph

The last section showed how to represent any Complex Artifacts as a rooted DAG of SKOnodes, the aim of this section is the creation of a single structure from several Complex Artifacts.

There are two main ways in which multiple Complex Artifacts, already converted into rooted DAGs of SKOnodes, can be made into a single structure:

1. *Sharing of nodes*: if two Artifacts share some data (for example, a piece of text, a table or an image), a single SKOnode can be created to represent the shared data and linked to by the nodes from both Artifacts.
   Thus, the shared SKOnode effectively joins both Artifacts and the resulting structure is a SKOnode DAG.
2. *Creation of a common higher-level node*: with no shared data between two Artifacts, a new SKOnode can be created and linked to the root of two or more SKOnode DAGs.
   Thus, higher-level SKOnode effectively joins the Artifacts in a single SKOnode DAG.

Applying these joining procedures with the lemma (2) leads us to the following lemma:

If U represents a set of Complex Artifacts {A1, A2, ..., An} then for all U a G exists such that G is a SKOnode DAG and it includes the same data as U **(3)**

As an example, of the previous consider the following image:

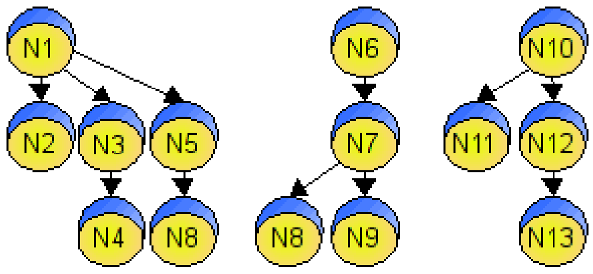

Fig. 4. Three separate Complex Artifacts represented as SKOnode rooted DAGs

The three Complex Artifacts from the previous figure can be joined into a single DAG by:

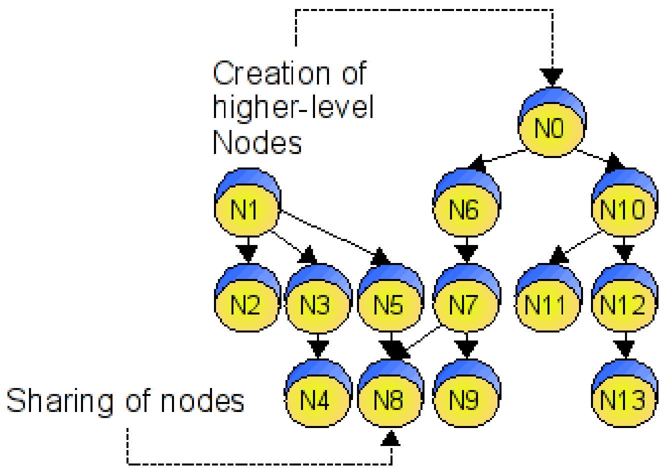

Fig. 5. Three separate Complex Artifacts joined into a single SKOnode DAGs

The SKOnode level is then able to represent all the data from an arbitrarily large set of Complex Artifacts into a Universal Data Graph which contains all the information of the original Complex Artifact Set as a SKOnodes related by the “Part of” relation.

2 The Knowledge Representation Level

The second level in this approach is the Knowledge Level and it is built on top of the, previously defined, Data Level. Knowledge represents the understanding or awareness gained through data as opposed to the data itself, which just represents facts and descriptions.

One of the objectives of this level is to organize the data nodes into Knowledge Objects, and it does so by picking a node from the Universal Graph Structure, making it the root of a rooted DAG and adding global metadata on top of it. As such, while the Data Level may be a DAG, each Knowledge Level structure has only their personal rooted DAG from the universal DAG.

However, besides the organization of data nodes, the objectives of the Knowledge Level also include to provide the tools to enable evolution, collaboration and the composition of existing Knowledge into new propositions of Knowledge.

The chapter will first give the definition of the basic structure of this level, the SKO, along with its components, and then it will also introduce its Serialization Structure.

2.1 SKO Definition

This section will define the SKOs or Scientific Knowledge Objects which is the basic structure to be used for organization at this level.

The following graphical representation gives a basic idea of the components of the SKO structure:

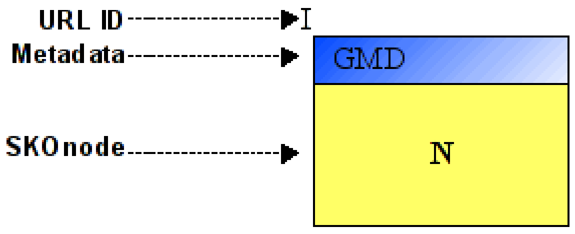

Fig. 6. Graphical representation of a SKO

More specifically, a SKO S is defined as the following ordered 5-tuple:

$$S= <URL, GMD, N>. \qquad (4)$$

Where:

- *URL*: provides a unique identifier for the SKO as a whole.
- *GMD*: Global Metadata contains SKO related information. The GMD is defined as a set of attributes, in the same way as the MD component from SKOnodes was.

  Additional requirements for the SKO's GMD include:
  - *State*: a *unique* attribute with the Label or Name “State” *must* always be defined. Furthermore, its value must be either “Gas”, “Liquid” or “Solid”. More information about State and its uses will be given later on Chapter 5.

    - *Serialization Structure:* a unique attribute with Label or Name "Serialization Structure" *must* be defined and its value *must* also be defined.
- *N*: is a SKOnode, as defined in the previous chapter. This node is taken as the root of a rooted DAG of SKOnodes from which the SKO will be built.

Depending on the decided implementation of the previous structure, all the other components that come after GMD can be chosen to be implemented as attributes contained in GMD.

Some basic SKO examples include papers, books, presentations, photo albums, recordings among other Complex Artifacts.

2.2 SKO Serialization Structure

Based on the previous definitions, this section will introduce the SKO Serialization Structure and its relation to the previously treated SKOnode Semantic Structure.

Continuing with the example shown on figure 3 from section 1.2, we will assume that we have a document with a clearly defined Title, Chapters and Paragraphs and we want to represent that as a SKO. By making S = <URL, GMD, Nr> a SKO is now wrapped around the whole structure and adds Global Metadata (GMD) to it.

The following image is an example of the result:

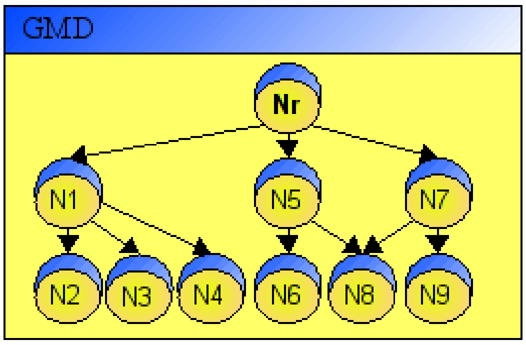

Fig. 7. Conversion of the data tree of Figure 3 into a SKO, the light blue box represents the SKO's GMD while the yellow box contains an extended view of the node pointed at by the SKO and its descendants

This SKO keeps its semantic from the SKOnodes intact, as each link still represents the "Part of" relation.

However if presented "as is" to a human user, its branching nature may make it difficult to read and understand. The solution for this problem is the definition of something that would turn the rooted Graph from the previous image into a list, like the following image:

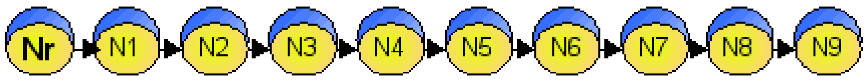

Fig. 8. Serialization structure for the previous document example

Just like in the previous chapter, Nr represents the document's Root(containing the Title and other general information), N1, N5 and N7 represent the introduction of each chapter, while the bottom nodes represent its paragraphs. Thanks to the Serialization Structure the document is now ordered in a much more understandable way to human users. We will call this list-like structure the Serialization Structure (SS) of a SKO S. This SKO Serialization Structure is defined as the ordered n-tuple:

$$SS= <N1, N2, ..., Nn>. \tag{5}$$

Where each of the components of the n-tuple represents a SKOnode that is either N, the root node pointed by the SKO, or one of his descendants.

The most common use for the Serialization Structure is to “assemble” the SKO into a format that is much more understandable to humans.

2.3 Composing SKOs

One of the main objectives of this level is to help the proposition of knowledge based on previously existing knowledge. An example of this is the creation of a book that arranges, presents and discuses several previously existing Artifacts.

To achieve this is necessary to define a way of linking SKOs in such way that a child SKO C can be said to be part of parent SKO P. This linking is done in the two levels we currently presented.

Data Level Linking. To create a data-level link between the SKOnodes from two SKOs its only necessary that one SKOnodes belonging to the parent SKO links to the root of the child SKO. An example of this procedure is shown in the next figure.

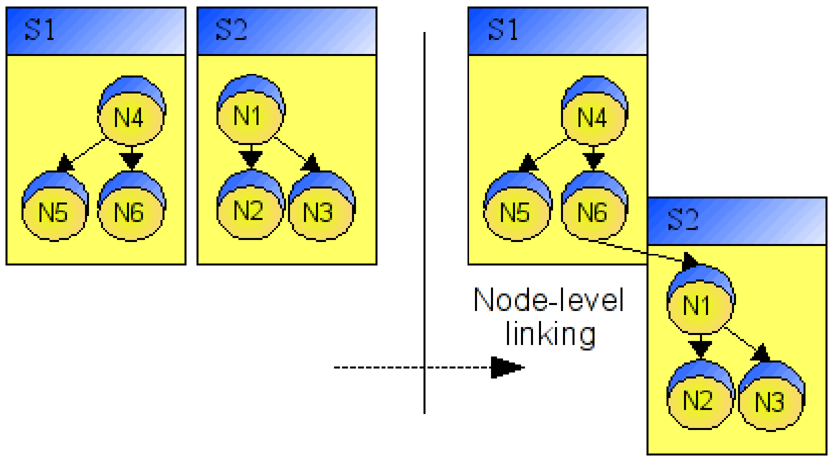

Fig. 9. Example of a Link of the SKOnodes belonging to two different SKOs

Note that a data-level link between two SKOs only ensures that the data is shared between SKOs but it doesn't say anything about how this data is ordered, its context or the knowledge that this data may propose.

Knowledge Level Linking. To create a knowledge-level link between two SKOs it is first necessary that a data-level link exists. Once that requirement is complied, a "Composes" SKOlink from the Parent SKO to the child SKO can be created to establish the knowledge-level link. For more information about SKOlinks please refer to chapter 4.3.

An example of this procedure is shown in the next figure.

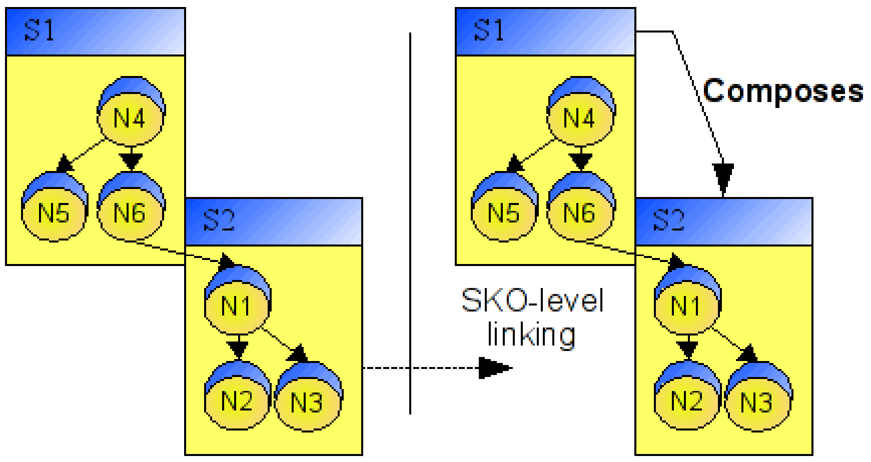

Fig. 10. Example of the composition of two SKOs

A Knowledge-level link ensures that the Child SKO is presented as part of the Parent SKO *exactly* as it was presented individually. For this reason, when creating a Serialization Structure for a SKO that composes other SKOs it is mandatory to follow the same Serialization Structure of the original children SKOs.

If at any moment this requirement is deemed too strict and the data from the child SKO wants to be used on a different way than it was in the original SKO, its only necessary to break the knowledge-level link (deleting the SKOlink) while still keeping the data-level link.

3 The Collection Representation Level

The third and final level of organization in this approach is the Collection Level, built on top of both the, previously defined, Data and Knowledge Levels. Collections here represent a grouping of data and/or knowledge that have some shared significance with each other.

One of the objectives of this level is to group related structures. It does so by defining a set of conditions that selects a set of objects and adding global metadata on top of that selection.

While these collections also organize data and knowledge as the previously defined structures, the weaker semantic relation between its components makes them ideal for category-like uses like bookmarks and workspaces. Furthermore unlike the previously defined levels, resolving which objects are included is resolved intensively (by conditions) and dynamically (at run-time, for example).

The chapter will first give the definition of the basic structure of this level, the SKOset, along with its components and considerations.

3.1 SKOset definition

This section will define the SKOset which is the basic structure to be used for organization at this level.

The following graphical representation gives a basic idea of the components of the SKOset structure:

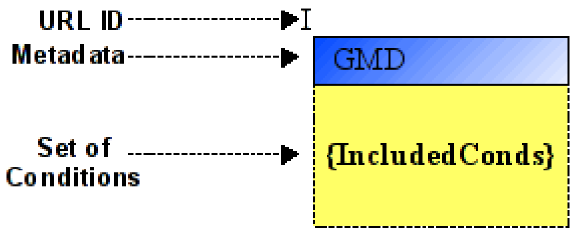

Fig. 11. Graphical representation of a SKO

More specifically, a SKOset S is defined as the following ordered triplet:

$$Sset = \langle URL, GMD, \{IncludedConds\}\rangle \quad (\mathbf{6})$$

Where:

- *URL*: provides a unique identifier for the SKOset as a whole.
- *GMD*: Global Metadata contains SKOset related information. The GMD is defined as a set of attributes, in the same way as the metadata component from SKOnodes and SKOs were.
  As an additional requirement for the SKO's GMD:
  - *State*: a *unique* attribute with the Label or Name “State” *must* always be defined. Furthermore, its value must be either “Gas”, “Liquid” or “Solid”. More information about State and its uses will be given later on Chapter 5.
- *{IncludedConds}*: is a possibly empty (0...n) set of conditions. These conditions can be used to pick sets of either SKOnodes, SKOs or SKOsets(albeit with cycle prevention considerations in the last one).

Depending on the decided implementation of the previous structure, all the other components that come after GMD can be chosen to be implemented as attributes contained in GMD.

Furthermore, please note that the SKOset is the first structure in this document that has a recursive definition, meaning that SKOsets themselves may be a part of a SKOset. This composability helps the Collection Level to consolidate its position as the last level of organization in this approach.

Some basic examples of SKOsets are personal bookmarks, workspaces, categories and even search result sets.

4 Complementary Definitions

With the basic structures of the three-level organization approach already covered at the previous chapters, this chapter will focus on describing some other objects and structures that complement the approach.

Specifically, the chapter starts dealing with the metadata and attribute-definition to later focus on links between the defined structures. Structures related to Authors and types of SKO-based structures (SKOtypes) will also be briefly touched but their definition will be left for future works

4.1 Metadata and Attributes

Metadata and attributes were briefly mentioned as part of the structure definitions from the last three chapters. In particular, MD from the SKOset and both GMD from the SKO and the SKOset are defined as a set of attributes:

$$MD = GMD = \{A\} \quad (\mathbf{7})$$

Where A represent attributes. Attributes, in turn, are defined as:

$$A = <URL, L, T, V, O, D> \quad (\mathbf{8})$$

Where:

- *URL*: provides an identifier for the attribute.
- *L*: attribute label in Natural Language, implemented as a string.
- *T*: attribute type, this is implemented as a closed list of options. This component determines the acceptable range and format of the accepted values for V. This component may be used to denote:
    - *Simple types*: like integer, float, etc, or
    - *Complex types*: like date/time and user defined binary-based ones, with specific formats expected from them.
- *V*: attribute value, limited by T and implemented as a string or binary.
- *O*: possibly empty offset, this is used on some attributes to specify the start of the specific portion of data to which the attribute refers to. Having Offset set to an empty or nil value, means that the attribute itself applies to the whole object and not to a specified part of it
- *D*: possibly empty duration, along with offset this specifies the exact part of the data to which the attribute refers to. Having Offset defined and Duration set to an empty or nil value means that the attribute itself applies from the part of the object specified by Offset until the end of that object.

Basic use of metadata includes format-specific attributes, which identify and carry information about the specific format in which the Physical Data is encoded; semantic information, which carries the subjects and meaning related to the Physical Data; and special attributes that may carry annotations, links to other sources or instructions on how to render the data, among others applications.

Metadata-linked Data. For text-based Artifacts like papers, items like Author, Title, and references are difficult to classify as exclusively belonging to the data or metadata category because they seemingly act as both in the following ways:

- *as data*: the author, title and references can be actually written *in* the text of the document and as such they form part of the data of the SKO.

- *as metadata*: author, title, etc, are in fact a classic example of metadata and form part of several metadata specifications.

As a solution to this situation, Metadata-linked Data defines a way to make certain values that are contained in specific metadata attributes to also appear in the data of the Artifact they represent. The exact specification of how this is achieved will be left for further works but an indication of it will be given here.

To implement Metadata-linked Data the definition of a suitable Markup language, that introduces these references to the object's metadata attributes from within the object's data, would be necessary.

For example, assume that we have the following attribute A = <URL, "main_topic", string, "Computer Science", 0, 0> within the metadata of an object which has a written text defined as its data. When the time to write the main subject of the text within the text itself comes, instead of writing "Computer Science" again, the author may choose to write "%main_topic%" instead. In this example the used Markup language prints the value of the attribute with its label included between the percentile signs instead of interpreting the string literally.

4.2 SKOlinks

Besides the "Part of" links, which were presented at the SKOnode definition, no relational object or link between structures has been treated up to this point.

However, the interactions which exist externally between the Artifacts and Authors and internally between these two groups themselves, also contain information about these structures that its deemed interesting to capture.

As such this section will introduce SKOlinks whose main purpose is representing the existing relations between all the previously defined structures.

The following graphical representation conveys the basic idea behind SKOlinks:

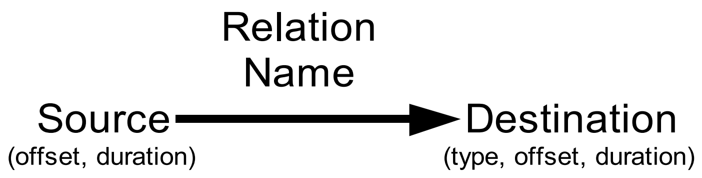

Fig. 12. Graphical representation of a SKOlink

More specifically, a SKOlink Link is defined as:

Link = <URL, Rname, Source, Soffset, Sduration, Destination, Dtype, Doffset, Dduration> . (**9**)

Where:

- *URL*: provides a unique identifier to the type.
- *Rname*: name of the relation, this is implemented as a closed list of options, for example "is related to", "references", "links to", etc. Furthermore, the available options depend on the type of object they are linking.
- *Source*: URL of the SKO, SKOnode, SKOset or Author that is the source of the link.
- *Soffset*: possibly empty, used to specify the start of the part of the source that starts the link.
- *Sduration*: possibly empty, used along with Soffset to specify the portion of the source that starts the link.
- *Destination*: URL of the SKO, SKOnode, SKOset or Author that is the destination of the link.
- *Dtype*: deals with the multi-version aspect of the link. In the current definition it can take either the value:
  - *"Current"*: meaning that the link refers to the latest accepted version of the object linked,
  - *"Newest"*: meaning that the link refers to the newest version of the object linked, or
  - *"Strict"*: meaning that the link refers to the exact object linked.
- *Doffset*: possibly empty, used to specify the start of the portion of the destination that is being linked to.
- *Dduration*: possibly empty, used along with Doffset to specify the portion of the destination that is being linked to.

SKOlinks are what are used, for example, on a web Artifact to link to another web Artifact, or in text Artifacts to reference other Artifact. Another less visible use of SKOlinks is to create semantic relations between SKOnodes, SKOs, SKOsets and Authors. The following table is an example of possible SKOlinks:

Table 2. Examples of possible predefined Rname values based on the structures it links

Source type	Destination type	Rname	Purpose
SKO	SKO	"reference"	Reference from one work to another
SKO	SKOset	"is included in"	State that a certain work is included in a certain collection
Author	SKO	"is collaborator of"	State that a person has collaborated for the creation of a certain work
Author	Author	"is colleague of"	State that a certain collaboration exists between two persons

A complete listing of all the relations and its details for each of the objects will be made available at future works.

5 The State Dimension

Just like in the physical world the same matter has very different properties and behaviors depending on the state it is in, the general properties and allowed operations of each of the previously defined three main structures of the approach vary greatly depending on the value of their State property.

The definition of this State dimension and its interaction with the previously defined structures is the main objective of this chapter.

5.1 Purpose of each State

This section will introduce each of the three proposed states; Gas, Liquid and Solid, along with their most important properties and their overall purpose.

Table 3. Summary of properties of each of the three States

Property/State	Gas State	Liquid State	Solid State
Development Level	Early	Tentative	Finalized
Main Purpose	Internal development	Feedback and partial dissemination	Conventional publishing and mass dissemination
Target	Immediate work-group	Development partners	General public or target audience
Modification Interval	Frequent and unordered	Correction batches based on feedback	No modifications
Contributions	Loosely tracked	Strictly tracked	No modifications

Each of these States will be discussed with more detail in the following subsections.

The Gas State. Structures at the Gas state are mainly used as the starting point for changes and evolution. The following list contains the main characteristics of structures in the Gas state:

- *Modifications overwrite*: each of the modifications done to the objects in the Gas state can partially or fully overwrite the previous existing information.
- *Loosely tracked modifications*: in this state it is expected that authors introduce frequent and significant changes, many times overwriting previously existing information. This makes difficult the keeping of a fine-grained detail of the changes introduced and the level of collaboration from the authors.

- *Authorship*: since modifications are not strictly tracked and information can be overwritten, Authorship for a Gas object cannot clearly be determined by the system. Authors themselves are charged to define their participation level at the creation of the object.
- *Maturity of information*: the information from Gas object is deemed as preliminary. As such, Gas objects are not generally considered worthy of being cited or referenced and carry no significant impact for the Authors.
- *For development or close-knit collaboration*: the preliminary evolving ideas, the frequent and loosely tracked modifications and authors having to define collaboration themselves, all of these factors point to a very reduced number of trusted collaborators working on the Gas State object.

The current equivalent to Gas SKOs would be the work-in-progress documents that are written by individuals or relatively small teams. Because of its preliminary nature, they are frequently treated with relative secrecy (particularly on research or business cases). Concrete examples include regular MS Word or LaTeX files and on-line collaborative editing tools like Google Docs.

The Liquid State. Structures at the Liquid State are still evolving and being modified by their original creators, however this process is a lot more ordered than in the Gas State. Liquid objects can also be opened for collaboration and discussion within a group of people, mirroring some sort of closed (or open depending on the owner) Beta Test from the Software Development world. The specific characteristics of structures at the Liquid State are the following:

- *Modifications version*: on the Liquid State each of the modifications introduced to the object creates a new version of it. This is fundamentally different to the Gas State modification where information is overwritten.
- *Strictly tracked modifications*: the modifications are expected in this state are normally batch of corrections and medium/small additions. As such, not only each modification can be is attributed to a specific author and a specific date/ time but these modifications are also reversible.
- *Authorship*: thanks to the strict tracking of modifications, it is possible accurately determine the collaboration level of the Authors that participated on the creation and modification of the Liquid object.
  Additionally in the Liquid State objects, it is possible for the Authorship to gradually change in time as new collaborations are added.
- *Maturity of the Information*: the information from Liquid objects is normally considered to be of Draft or Request-for-Comment quality and is expected to contain relatively stable content. As such, Liquid objects can be cited and quoted and may even influence the Author's standing in his community
- *For partial dissemination and feedback*: the previous characteristics enable the Liquid objects to better coordinate and keep track of the collaborations and authors which, in turn, enables them to be opened to a larger group of people for early dissemination or for obtaining feedback.

While we already introduced work-in-progress documents as an existing example of Gas state objects and the next subsection will deal with the Solid objects which are very similar to the current publications, there is no current existing example of the Liquid State objects described in this approach. The closest existing example are the widely used Software Repositories from the software development world and Wiki-like content managing approaches.

As such one of the main objectives and added value of this approach is the introduction the Liquid State objects as a bridge between currently existing and widely used Gas-like and Solid-like objects.

The Solid State. Structures at the Solid State have finished their evolution and are ready to be disseminated to the world. The specific characteristics of structures at the Solid State are the following:

- *No Modifications allowed*: no modifications are allowed to be made to a Solid State object, nor direct versioning of it is allowed. As such there are no modifications to track in this state.
- *Authorship*: the authorship of a Solid object remains fixed and unequivocally defined from the moment of its creation.
- *Maturity of the Information*: the information from Solid objects is normally considered to have the highest grade of maturity and treated as of being of release quality. The previous, along with the general stability offered makes the Solid State object the most adequate for citing, quoting and influencing the Author's standing and prestige.
- *For publishing and general dissemination*: the “rigidity” of the Solid State is especially appropriate for public releases and dissemination.

The current publication system works exclusively on these Solid objects, they are the most common and visible types of objects.

5.2 State Change Operations

State change operations are used to make any of the previously defined structures create a copy of itself on a different State. The following figure illustrates the different states and the transitions that are possible between them:

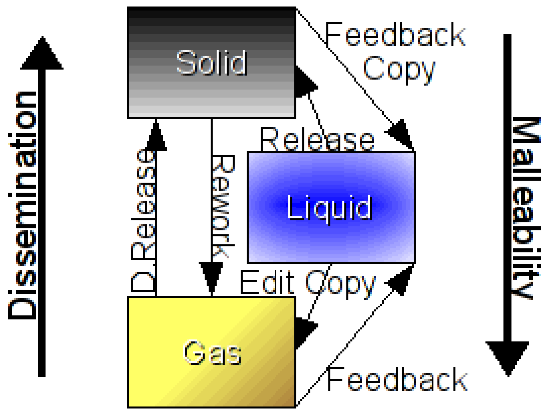

Fig. 13. State transition diagram for SKOnodes, SKOs and SKOsets

Based on the previous figure the following operations can be identified:

- *Gas to Liquid Transition*: used normally to open a draft or early work to a bigger amount of people and to obtain feedback, comments and ideas from them.
- *Liquid to Solid Transition*: used to publish or make a release of a work that has been previously opened for feedback, discussed and probably collaborated upon by a group of people.
- *Gas to Solid Transition*: used to publish or make a release of a work directly without going through an open feedback phase.
- *Liquid to Gas Transition*: used to create a directly editable copy of a work that is currently under discussion and collaboration from others. The objective of such copies is normally to introduce major changes on the previously mentioned work.
- *Solid to Liquid Transition*: used to create a versionable copy of a currently published work. The objective of such copies is to collaboratively introduce small refinements to the previously mentioned work.
- *Solid to Gas Transition:* used to create a directly editable copy of a currently published work. The objective of such copies is normally to introduce major changes on the previously mentioned work.

5.3 Structure Specific Considerations

This section will detail the State change operations for each of the introduced structures (SKOnodes, SKOs and SKOsets).

In general each transition is implemented as the creation of a new object which is the copy of original structure in the target State. However the process has its particularities involved on each case, which will be detailed in the next subsections.

SKOnode State Transitions. SKOnodes are the only structure that points directly to data and are affected if this data is changed.

On every State Transition a new SKOnode will be generated that is initially a copy of the original SKOnode (on transition duplication). However the data itself will *not* be copied or duplicated during the State Change operation. Instead the data will only be copied or duplicated when the operations applied to it are incompatible between the different SKOnodes that point to this data (on modification duplication).

The following figure exemplifies both cases.

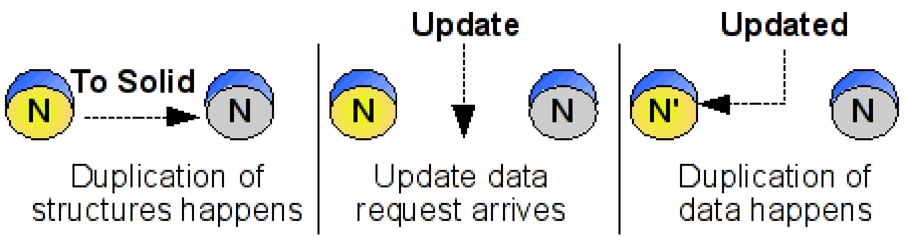
To Solid
N
N
Duplication of structures happens
Update
N
N
Update data request arrives
Updated
N'
N
Duplication of data happens

Fig. 14. SKOnode duplication and data duplication example

The previous image is split in three significant moments:

1. Note that on the State transition only the structure itself (ie. Metadata and other components) is copied but the data itself is not copied (both nodes point to the same unchanged data N)
2. When an update request for the data pointed by the two nodes is detected a conflict arises because the Gas SKOnode allows the modification but the Solid SKOnode does not.
3. The conflict is resolved by the auto duplication of the data, the Solid SKOnode is left pointing to the original unmodified data while the Gas SKOnode points to the changed data.

SKO State Transitions. SKO transitions are implemented normally, by creating a duplicate of the original SKO.

However, SKOs refer to SKOnodes instead of directly pointing to data, the particularity brought up by this fact is that SKOnodes also have their own State which may or may not be equal to the SKO that contains it.

In general, it is not allowed for any of the SKOnodes pointed by the SKO's Serialization Structure (that are the ones directly used by it) to be on la less restrictive state than the SKO itself. More specifically:

- When a SKO changes to a Liquid State, all the SKOnodes in its Serialization Structure that are on the Gas State will also be transitioned to the Liquid State.
- When a SKO changes to a Solid State, all the SKOnodes in its Serialization Structure that are on either the Gas or Liquid State will also be transitioned to the Solid State.

The following image gives an example of this process:

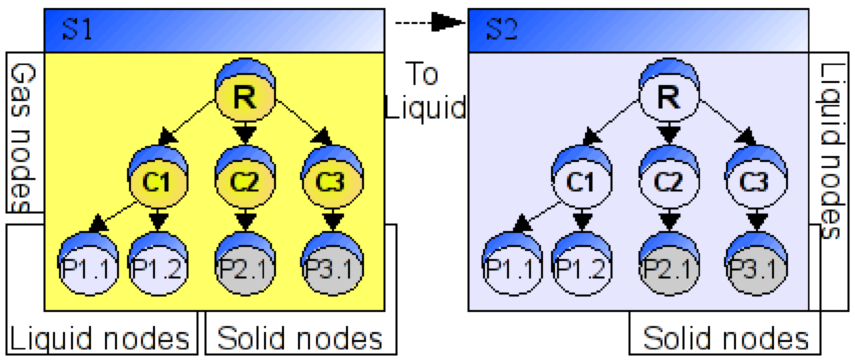

Fig. 15. Example of a SKO state transition

This precaution is in place to make sure that operations that are forbidden on a given State are not introduced into the Knowledge Level(SKOs) through the Data Level(SKOnodes).

On the other hand there is not any issue with SKOnodes being on a more restrictive State than its SKO. More specifically as SKOs in the Gas State may point to SKOnodes in both the Liquid and Solid States and SKOs in the Liquid State, may point to SKOnodes in the Solid State.

More information about operations and whether they are allowed or forbidden for each State can be found on the next chapter.

SKOset State Transitions. While SKOs determine their elements statically and by extensive definitions, SKOsets determine which elements are included in them by conditions that can be checked dynamically. Furthermore, modifications to a SKOsets are done on these conditions rather on the content included itself so a much higher level of independence exist between the structure and its contents.

This particularity makes the safeguards against disallowed operations put up for the SKOs meaningless for the SKOsets. If any component suddenly changes and no longer complies with the conditions set by the SKOset, they will be automatically excluded from the SKOset. The following image gives an example of this process:

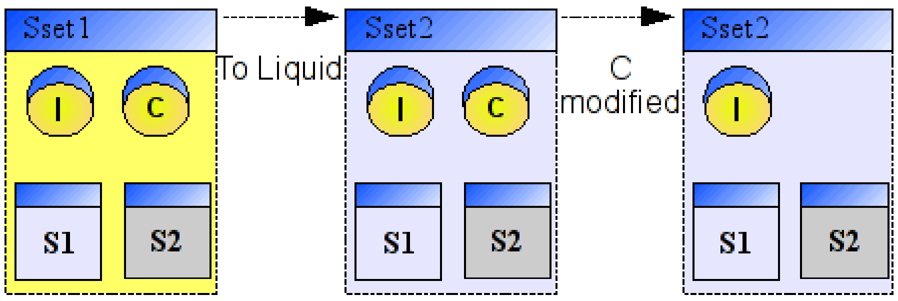

Fig. 16. State transition of a SKOset and auto exclusion of one of its members

The previous image is split in three significant moments:

1. Note that even though the SKOset changes its state, all the structures it includes remain on their same original state.
2. One of these included structures is changed and this change makes the object non-compliant with the conditions defined in the SKOset structure.
3. Thus, it is automatically excluded from the SKOset.

This behavior is consistent with the view of SKOsets as Categories or “bag of things”, rather than being charged with representing Data and Artifacts like the previous structures.

6 Basic Operations

With all the structures and their States already defined at the previous chapters, this chapter will focus on the definition of the basic operations for these structures.

While the current State of the structures and the sharing of their multiple resources will be extensively considered on such definitions, the access and permission issues will not be. As such, unless explicitly noted, for the purpose of all the operation definitions contained in this chapter, it will be always assumed that all the necessary author/user permissions are in place for the operation to happen.

The following table introduces the four Basic Operations for the SKOs, SKOnodes and SKOsets along with how their Current State affects them:

Table 4. Relation between the State of the SKO, SKOnode or SKOset and their allowed basic operations

State/Op	Create	Read	Update	Delete
Gas	Yes	Yes	Overwrites	Yes
Liquid	Yes	Yes	Versions	No
Solid	Yes	Yes	No	No

The justification and details for the values in the previous table will be given in the next sections.

6.1 Create

The Create operation introduces new instances of structures to the system.

Objects can be created either on:

- *Gas State*: if they are expected to go through significant changes after their introduction to the system.
- *Liquid State*: if their main objective is to be obtain feedback and maybe collaboration from others.
- *Solid State*: if their main objective is dissemination and some limited community-related functionalities.

The Create operation, however, is divided in two sub-operations: Create New and Create Duplicate, which will be detailed in the following subsections along with other considerations.

Create New. This operation creates a new URL and initializes the structure's values according to the passed arguments.

Function Examples:

```
URL = create_new_skonode(Gas, MD, PD, LinkedNs)
URL = create_new_sko(Solid, GMD, N)
URL = create_new_skoset(Gas, GMD,IncludedConds)
```

Create Duplicate or Fork. This operation creates a new URL for a duplicate of an existing structure.

Unlike Versioning this new object is considered to be completely detached from the original. These duplicates are used generally for forks or copies that intend to extend or deviate from the original.

If we assume that url_source is URL that points to the object that wants to be duplicated, then Create Duplicate can be implemented by:

1. *New URL*: obtain an URL url_duplicate for the new object
2. *Structure Copy*: follow url_source and copy all the structure's content into url_duplicate
3. *Outgoing links copy*: copy all the SKOlinks that have url_source as their source, replacing in each copy url_source with url_duplicate
4. *Declare the duplication*: create a new “Duplicated from” SKOlink from url_source to url_duplicate.

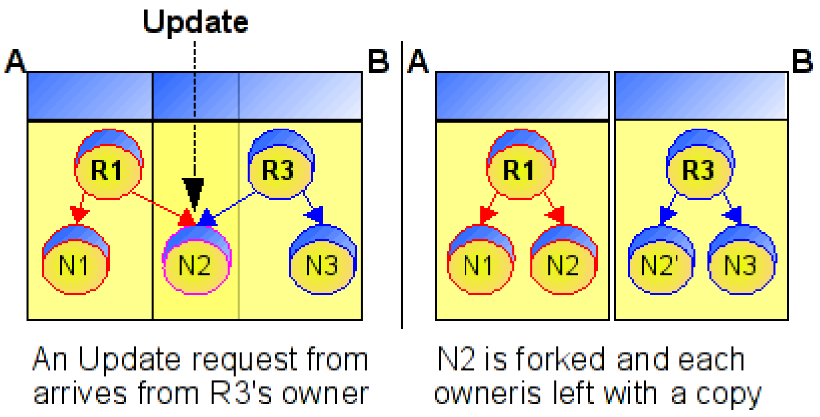
Update
A
B
A
B
R1
R3
N1
N2
N3
R1
R3
N1
N2
N2'
N3
An Update request from arrives from R3's owner
N2 is forked and each owneris left with a copy

Fig. 17. Example of Forking of a SKOnode N2 that was previously shared by two different Author. After the fork, each Author is left with their own exclusive node.

Function Examples:

```
URL = create_duplicate_sko(Gas, url_source_sko)
URL = create_duplicate_skoset(Liquid, url_source_set)
```

6.2 Read

The Read operation is used to retrieve information from the existing structures.

This operation is allowed regardless of the current State of the object and its only dependant on access-control rights.

Function Examples:

```
SKOnode = read_skonode(url_node)
SKO = read_sko(url_sko)
SKOset = read_skoset(url_set)
```

6.3 Update

The Update operation introduces changes to already existing structures.
The following list specifies the behavior of the Update operation depending on the State of the target Structure:

- *Gas State*: Update is carried out as an Overwriting Update.
- *Liquid State*: Update is carried out as an Versioning Update.
- *Solid State*: Update is not allowed.

These two types of Update operations will be detailed in the following subsections along with other considerations.

Overwriting Update. This is the Update operation that corresponds to the Gas State. As its name implies the Overwriting Update simply replaces the existing Structure information with the ones passed by the arguments.

Function Examples:

```
update_sko(url_source_node, GMD, PD, LinkedNs)
update_skoset(url_source_node, GMD, IncludedConds)
```

Versioning Update. This is the Update operation that corresponds to the Liquid State.

Instead of overwriting, this Update operation preserves the original and applies the update to a copy of it but, unlike Forking, this new object is considered to be another configuration or development stage of the original object. These versions are generally used as collaboration and development tools aimed at producing a single final result.

If we assume that url_source is the URL that points to the object that wants to be versioned, then the Versioning Update can be implemented by:

1. *New URL*: obtain an URL url_duplicate for the new object
2. *Structure Copy*: follow url_source and copy all the structure's content into url_duplicate
3. *Outgoing links copy*: copy all the SKOlinks that have url_source as their source, replacing in each copy url_source with url_duplicate
4. *Apply changes*: apply the modifications carried by the update operation to the newly obtained copy.
5. *Declare the versioning*: the existence of new version is registered in the corresponding version control structure and a new "Versioned from" SKOlink from url_source to url_duplicate is created.

Note that the following Function examples are exactly the same as the ones from the Overwriting Update operation, which means that the current State of the target object is what decides if an Overwriting or Versioning Update is done.

Function Examples:

```
update_sko(url_source_node, GMD, PD, LinkedNs)
update_skoset(url_source_node, GMD, IncludedConds)
```

Update Considerations About External Data Modifications. Besides the direct modification of the Structures, indirect(outside of the control of the system) Data Updates may happen to the data pointed by SKOnodes and these have to be detected and either allowed or denied depending on the State of the SKOnodes that point to it.

To be able to detect and react properly to these external changes it is necessary to store an internal copy of the Data pointed by the SKOnodes. Assuming that PD represents the Physical Data pointer to the original data and PDC represents such Physical Data Copy internal to the system, once a difference is found between the data pointed by PD and PDC there are three actions that the system needs to consider:

- *Allow data modification*: if all the SKOnodes pointing to the data allow the requested modification, the modification is allowed by refreshing PDC from the current PD.
- *Deny data modification*: if all of the SKOnodes pointing to the data deny requested modification, the modification is denied by making PD point to PDC. This effectively detaches the SKOnodes from the external data source.
- *Auto-Fork or Auto-Version*: if some of the SKOnodes pointing to the data allow the requested modification and some of them deny it, then the user is asked to make a Fork or a Version. The SKOnodes that denied the modification will detach themselves from the external data source while the SKOnodes that allowed it will refresh their PDCs.

6.4 Delete

The Delete operation is used to destroy the information and free the memory used by existing structures.

This operation is only allowed while on the Gas State as the system assumes that once something has been published or collaborated upon, it can no longer be retracted or make as it never existed.

Function Examples:

```
delete_skonode(url_node)
delete_sko(url_sko)
```

7 Final Words

This document has introduced, SKOs, SKOnodes, SKOsets and other related structures along with its properties and operations. However all the information contained here should be treated only as the first version of such specification, as additional details, specifications and refinements still need to be added in future versions.

Following the terminology set on this document, this version should be considered only as a Gas version of the final document.

8 Acknowledgements

This work is part of the Liquid Publications Project[3]. The LiquidPub project acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 213360.

References

[1] Liquid Publication: Innovating the Scientific Knowledge Object Lifecycle. Small or medium-scale focused research project (STREP) Full proposal. Set 2007.
[2] Casati, Giunchiglia, Marchese. Publish and perish: why the current publication and review model is killing research and wasting your money. ACM Ubiquity. Nov 2006
[3] http://project.liquidpub.org/