Title (alternate): XooML2. A practical, modular, distributed, application-independent, application-accommodating, infrastructure-neutral, simple (as can be) XML representation of the essential structures we need to access, make sense of and take control of our information.

Authors: William Jones, Lizhang Sun

Readers: The writing here is conversational, the scholarly references are minimal. The intention to inform and to spark an on-going discussion concerning the future of our information in which you too, Dear Readers, can also participate (<ref to blog>).

I. Introduction

This is to motivate and describe XooML2.

Motivation comes, on the one side, in the pain of a present problem: Our information is scattered, ever more so and paradoxically, by the very devices and applications we use to manage this information.

Motivation comes, on the other side, in the potential of a better situation: Our information – content and ***structure*** -- has an integrated, unified existence independent of any particular device or application. Just as we might use a diversity of tools for the tending of a physical garden, we apply a diversity of tools ("the right tool for the right job") as we tend our growing garden of personal information.

We note that the motivation of pain may not suffice for those of us who are coping nicely (or at least "OK") with a current situation. But many more of us, if we use our imaginations even a little, are motivated by the potential power of a future in which we are able to access, "data mine", make sense of and use our information much more effectively than today. This is a future in which we have access to and control of our information – structure as well as content.

If the "why" of motivation is, for now at least, accepted, we can focus on the "what" and "how" of XooML2's description. What is XooML2? What is its approach? And how can it help? Additional "whys" will also be addressed. Why, for example, the XooML ("1" and "2") approach and not others?

XooML2 (draft)

Like its predecessor, XooML[1], XooML2 uses XML to give tool independent, but "tool accommodating" expression to a basic, ubiquitous unit of information structure. We call this unit a *grouping item.* We also refer to the structure of this unit as a "noodle" as in "node + outgoing links".

Grouping items are called by different names in different information management applications. Evernote, for example, provides for "notebooks", "notebook stacks" and "tags". MS OneNote, an alternate note-taking application, provides for "notebooks", "section groups", "sections", "pages" and "sub-pages". A file system like that used in MS Windows or on the Macintosh provides for directories or "folders". Even indexing and search utilities such as Lucene can be seen to create a kind of grouping item in the form of a term + links to documents in which the term has been found or to which the term is otherwise associated.

Grouping items differ from one another in many respects such as the manner of their creation and the manner of and restrictions in their use. For example, folders are created by people (except, of course, for all the other folders that are created by applications in the course of their installation and use). Index terms and their lists of associated documents, on the other hand, are created by an indexing utility (though people can exert indirect control over the indexing process through choices made with respect to indexing components such as the word stemmer and the word separator). For examples of use, consider tags in Evernote vs. tags in OneNote. Evernote tags can be structured into a hierarchy. Tags in OneNote, on the other hand, can each be given a font, a color and an icon but cannot be structured into a hierarchy.

Notwithstanding these and other notable differences, all grouping items have in common a basic "noodle" structure: A grouping item is a node + outgoing links. Links point "directly" (i.e., via an address such as a URI that, in most cases is quickly and unambiguously resolved) to other information items including, when the application allows, to other grouping items. A folder points to its subfolders. An Evernote tag can point to other Evernote tags. The node of a grouping item can, in turn be addressed by other grouping items (where the application allows). The grouping item has an address. The grouping item has a name.

[1] XooML (pronounced "zoom'l") stands for Cross (**X**) T**oo**l **M**ark-up **L**anguage. An earlier version of the XooML schema is in use in several separate prototyping tools, including Planz and QuickCapture. For a complete definition of this earlier version of XooML see kftf.ischool.washington.edu/XMLschema/0.41/XooML.xsd. To try out these prototyping tools visit: http://kftf.ischool.washington.edu/planner_index.htm.

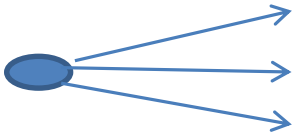

Figure 1. The explicit structure of a grouping item is a "noodle" (node + outgoing links).

The remainder of this paper divides into sections addressing the following questions:

1. **Why the grouping item as the basic unit of structure?** Why not instead, for example, the more atomic proposition as realized through a subject-predicate-object assertion of RDF?
2. **What is XooML approach?** Once a "noodle" (node + outgoing links) metadata structure has been extracted for a given grouping item, how can this metadata be used?
3. **What is XooML2 (and why)?** We consider lessons learned from a previous version of XooML and its use in several different tools.
4. **Conclusion.** Our information, integrated, under our control, for a lifetime of use. How might the vision really, realistically, be realized?

But first, …

Some terminology

Some terms are used throughout this paper in a certain way according to the following informal definitions:

- **Information item**. The chunk of information returned by the resolution ("de-referencing") of an address such as a URI. Examples include a file, a Web page and an email message.
- **Document**. Information presented to a person through a tool for viewing and manipulation. Information item plus tool for viewing and interaction = document.
- **Grouping item**. An information item whose primary purpose is to form a grouping of other information items. There is no clear boundary between grouping items and other information items. We might say that the grouping item is not a thing but rather a way of viewing things – information items. Any information item has internal structure and references (explicit or implicit) and so can be viewed as a grouping item.
- **Noodle**. An addressable (linkable, nameable) node + the outgoing links from this node (pointing to other nodes). The noodle is the basic, explicit structure of a grouping item.

II. Why the grouping item as the basic unit of structure?

Consider these examples:

- To decide which hotel to book, we make notes from the Web for several alternatives, placing these notes on a OneNote page. Alternatively, we make notes in Evernote, giving each the tag "hotels-Boston".
- We're working to complete a complicated application process that requires the completion of several forms. We seek first to place all forms in one place – perhaps in a folder or even as printouts on a physical desktop. We do this to gain a clearer sense of the effort involved and to be sure we have "everything in one place".
- In our search for the current version of a document, we navigate to a folder containing several versions, sort these by "last modified" date and then select the most recently modified of these.
- We don't have a direct address for a targeted Web page but know that a hyperlink to this Web page is the second of two that can be found in the "upper left-hand corner" of another Web page.
- We want to contribute to an email discussion. To do so, we want both to locate the most recent post and also to locate previous posts so that we can reflect their content in our response. If we're using Gmail, this grouping is already done for us. If we're using MS Outlook instead, we'll opt to see "related messages in this conversation" and then we'll be sure to sort to so that we reply-all to the most recent post.

The folder, the OneNote page, the Web page, the subject line used in common among the email messages of a conversation – even the physical desktop – are forms of a grouping item. In each case, the grouping item, including its set of associations to other information items, provides an important context for the activity at hand – whether this is selecting the right document, the right hotel room, the right hyperlink, responding to the most recent email post in a conversation or, more generally, getting a sense for and making sense of the information at hand.

Several observations can be made.

The grouping item has the explicit structure of a "noodle"

The information in each of a grouping item's links can include an explicit address (e.g., a URI) quickly resolved with minimal computation. Explicit structure can be contrasted with implicit or "latent" structure. We can acknowledge that there is lots of implicit structure out there waiting to be discovered. We may be able to detect, for example, consistencies and correlations in our choice of words or our selections in information items (e.g., when, where and followed by what?). Implicit or "latent" information structure is, ever more so, the target

of data mining and text analysis efforts. But discovery of implicit structure – whether done by us or a computer -- is computationally expensive and time-consuming. The explicit structure of a grouping item lets us move on to other stages of information processing where structure, once discovered, is used and re-used.

The distinction between latent or implicitly represented structure and the explicit structure of a grouping item is illustrated through the email conversation. Only a few years ago, email messages as accessed through an application like MS Outlook were not indexed. There was then no "subject" or thread id term that could serve as a grouping item to point quickly to all messages in a thread. We could sequentially search instead through a larger (often much larger) set of email messages using a subject term or message id as a search string. But this could take a long time to complete – many minutes or even a lunch-time.

In those olden days, many of us opted instead to make the extra effort to use email folders to achieve a grouping of the messages of a conversation or related to a subject theme. Now increasingly, we're opting to use search instead[2] In doing so, we use the terms of an index as a form of grouping item. [3]

The grouping item establishes a critical context for the information activity.

Even in the two examples above where focus is on a single link – to the right version of a document or to a sought-for web page – the selection could not be made absent a larger context that includes a representation of alternate links. The grouping item supports a stepwise navigation to the desired information that both simplifies the activity of finding the desired information and also helps us more easily determine the relevance of this information once found[4]. Context becomes more important in cases where the activity is less well defined and we are trying to "see what we have here".

The grouping item can also have implicit structure and emergent properties.

The grouping item has structure explicitly represented in its outgoing links (associations) and their addresses (URIs) pointing to other information items. At the same time, as we perceive these links (as mediated by the sights or possibly the sounds of a tool) these links may seem to relate to each other and to node-level information in ways less easily described explicitly. "Semi-groupings" may "pop out" as gestalts. Our perceptions of these is influenced, for

[2] See Whittaker, Matthews, Cerruti, Badenes, & Tang, (2011).

[3] The use of index terms as grouping items works beautifully most of the time but is subject to the basic limitations of an index as a reflection -- one step removed" -- from the reality of the actual messages. If the index is out of date, the grouping we see may be out of date as well.

[4] See Bergman, Whittaker, Sanderson, Nachmias, & Ramamoorthy, 2010; Jones, Phuwanartnurak, Gill, & Bruce, 2005; Teevan, Alvarado, Ackerman, & Karger, 2004.

example, by the relative location of links in a display or by display properties of their appearance such as their color, size, or shape.

In the spirit of a "spatial hypertext" we may manipulate our perceptions of these semi-groupings as, for example, when we place notes close to one another on a page in OneNote. Such a manipulation is often a first step towards a more explicit chunking of links under a finer grained partition of grouping items. A grouping of links to web pages related to a "term paper on search" might be, for example, further divided into sub-groupings of links to web pages for "Lucene", "faceted search", "stemming and word-breaking", etc. It is often the case that these subgroupings emerge and can then be made explicit only after an initial process of discovery that happens in the larger context of the initial grouping.[5]

The creation of grouping items and the naming of these is then an essential step in the *problem setting* (i.e., the definition of the problem space) that precedes *problem-solving.* Before we figure out which hotel is best meets our travel needs in Boston, we first create a "hotels in Boston" grouping item affording ready access to information concerning hotel alternatives. Schon (1984) defines problem setting as "a process in which, interactively, we *name* the things to which we will attend and *frame* the context in which we will attend to them." (p. 40).

Making use of a chemical metaphor we might call the grouping item a "molecular" unit of structure. In contrast, the link, association or proposition (such as those formed by the subject-predicate-object expressions of RDF) is an "atomic" unit of structure. The notion of emergence has implications for efforts such RDF that aim for a more atomic level of structural representation. Some properties of a link and its referent – such as "most recent version" – can only be confirmed in the larger context of a grouping item. Even more so, relationships and emergent groupings of links require the larger context of a grouping item. The grouping item is the shared subject for a collection of propositions or a collection of links pointing to information (e.g. "Hotels in Boston" or "things to do this weekend").

The "noodle" alone is not enough.

Decisions – even the selection of a specific link – depend upon a larger context. Structure is only partly described by the explicit links of a grouping item. The properties of the node and each of its outgoing links – for example, display properties such as size, shape, color, display location – may also help us to perceive additional structures. We can call these " gestalts". But then we realize that in our efforts towards a metadata representation of grouping items, a

[5] As a complement to the bottom-up process of chunking (or composition), new grouping items achieving a finer-grained partition are also generated through a top-down process of *decomposition*. We may do this, for example, when we break a larger project such as "Plan trip to Boston" into smaller tasks more appropriately placed on a checklist (e.g., "make the hotel reservation", "book the flight".

stark "noodle' representation of the grouping item as a node + outgoing links is not enough, we also need to provide room for tool-specific (more generally, "namespace-specific") properties both at the level of the node and each of its outgoing links.

How? How can we realize both a tool-independent "noodle" metadata representation of the explicit structure of a grouping item and also leave "tool accommodating" room for additional properties of the node and its outgoing links that are so critical to our efforts to understand and manage our information? And then, given this metadata representation, what can we do with it? These are topics for the next section.

III. What is the XooML approach?

We begin with a story of information integration and a look at a tasty layer cake of information management in which our information – content _and_ structure – is the connective middle layer between "surface" tools for information viewing and manipulation on the top-side and services for storage, security and sharing on a bottom layer.

But then it's time for more details. How is XML used to represent the structure of a grouping item? And how then is this "XooML" representation used?

A story about structure: Out of the application shadows and into the sunlight.

Let's start with story about structure. Once upon a time (that time is now), our information structures were hidden in and scattered across many different applications. But then a sequence of events occurred:

1. With focus on the grouping item, **we use tools to read** (using application-supported APIs) and represent (using XML) **structure** – both explicit and implicit. Structure is represented in a manner that is both tool-independent and "tool-accommodating". (More about the "how " of this later on).
2. Using this representation, **we are now able to compare our structures across different applications**. We notice, for example, that the structure of our tag hierarchy in Evernote is quite similar to the folder hierarchy we're using in our personal filing system. This structure is also similar to the structure we're sharing with our teammates via Dropbox. Also, this structure is not that different from the structure we've created in a team wiki.
3. Moreover, this structure is similar to the one that we'll use (more or less) in the headings and sub-headings of a final team report**. If only we could copy the structure**…. But maybe we can!
4. Along with the new XML representation of structure, we have access to a **growing set of tools for considering, critiquing, comparing and… copying our structures**. Tools are generally basic but very effective at helping us to see the "forest" in our structures. These tools help us to answer some basic questions. *How do our structures relate? Which are working well for us? Which are outdated, no longer in use and just getting in our way?*
5. **Different tools give us different views and support different features and different modes of working with and analyzing our structures**. Some of these tools come to us through the efforts of researchers who are asking many of these same questions (but with an interest in answers that can be applied more broadly). Tools are sometimes created "by the community" with minimal effort by simply wrapping existing open-source software in a layer to "speak XooML".
6. And then one day we realize that we are not content just to use our new tools to consider, critique, compare and copy our structures. **We also want to use our new tools to *change***

these structures and to write these changes back to in to the stores used by our applications. But how?....

7. Voila! We realize **we can usually *write* changes to the structure** of our grouping items (and so to structure more generally) working through the same APIs that we used to read grouping item structure in the first place.
8. And now a bit of magic: **Structures that once could only be accessed and worked with through a single application can now be accessed and worked with through lots of different tools**. There are even "work-alike" tools that give us the functionality of our original applications.
9. **We decide to review our original applications**, one by one. For one application, Application A, we say "I really like everything about this application. I have information with A and I think I'll leave it there. But I'm really happy I have so many new ways to work with this information."
10. But for another application, Application B, we say "I always liked the features of B for viewing and working with my information but I never liked the way B shared my information with strangers. Also, B is not so good at synchronization between my devices. Now that I have a B' with most of B's features for viewing and working with information, I think I'll move my information to Application C which is focused just on "back-end" issues of storage (and security, sharing, synchronization, etc.) If Application C doesn't work out, no worries. There are many other storage-service applications eager for my business."
11. Happily ever after, we continue to link our information together in all sorts of new and useful ways. We're willing to invest the time and effort to do this because **our information – structure and content – is at long last truly ours.** Onward to a lifelong pursuit of information integration!

Our structure freed from applications and made "first class" now forms the critical middle layer of 3 layer cake[6]:

- The top-layer is the **surface** or "frontend". We want tools in this layer to help us to view and work with our information. Everything from "red eye removal" (for photographs) to OCR and spell checking is in this layer.
- The bottom layer is for the **store** or "backend". Where do the bits go? How are they stored? What is the support for security, sharing and synchronization?
- The middle layer is for our information – content and, especially **structure.** Once our information is made into a separate layer we have much more freedom to mix and match the applications we use in the top layer and the services we use on the bottom layer.

[6] Note this is not the same as MVC.

Figure 2. A three-layer "cake" of information with our structure as the critical connecting middle layer between top-layer of features for interaction and a bottom layer of services for storage, sharing & synchronization.[7]

A confection? A concoction? Or can dreams of information integration come true? To assess these questions we need to consider more details of the XooML approach as described in the next sub-section.

How is XML used to represent the structure of a grouping item? And how then is this "XooML" representation used?

In the XooML approach, the structure – explicit and implicit – of a grouping item is represented as a fragment of XML metadata. The associated XML schema is depicted in Figure 3.

[7] From http://www.surbitonart.co.uk/acatalog/Sugarcraft_Catalogue_Metal_cake_stands_127.html.

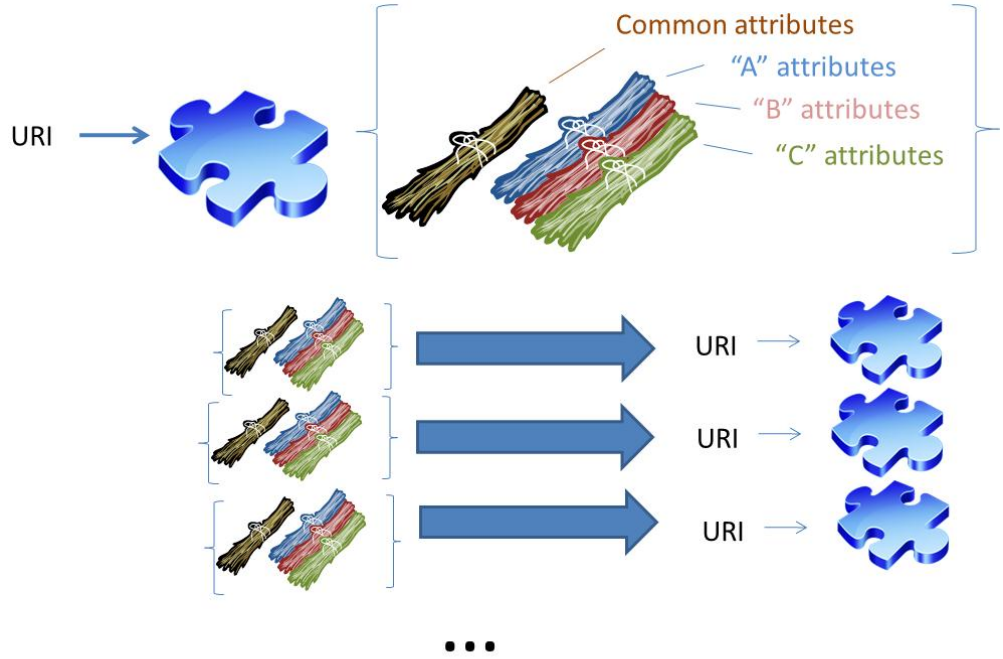

Figure 3. The schema for XooML provides for the description of a grouping item's structure as a fragment of XML metadata. A fragment consists of one or several bundles of attributes. One bundle is shared for common use by all tools that use the fragment. Other "namespace" bundles serve the needs of a single tool or a class of tools (e.g., all tools supporting Dublin Core). A fragment also includes zero or more associations that can themselves be modified by attribute bundles and that can point to other XooML fragments.

The explicit structure is represented through zero or more "associations" (links) each of which can optionally point to another fragment of metadata for another grouping item. At both the level of the fragment and the level of each of its associations there is a common "bundle" of attributes that should be used (properly) by all tools that work with the structure. At both levels, there is also provision for zero or more bundles of namespace- (tool-) specific attributes. (Each of these bundles is a separate sub-element uniquely defined through an "xmlns=" attribute assignment.)

With reference to Figure 3 and in the spirit of XML, a larger "document" – representing the structure of many grouping items -- can be constructed by a "XooML-speaking" tool like Planz through an assembly of XooML fragments (see Figure 4).

Figure 4. Planz generates a document-like view (in this example, of information relating to a home re-model) through an on-demand assembly of XML fragments – one per folder to be viewed.

The basic steps to render the document outline view on the right-side of Figure 4 are as follows:

1. Start with a URI pointing to a XooML fragment (and representing a grouping item such as the folder "House re-model". This becomes the "anchor" for the document to be rendered.
2. Get the XooML fragment and synchronize with the "reality" of the corresponding grouping item.
3. Update the XooML fragment as needed.
4. Use the (updated) XooML fragment to create a top-level view.
5. For each association, decide whether to "expand" i.e. to render its associated XooML fragment in the document for display. (Planz decides by inspecting a Planz-specific attribute named, "isCollapsed".) If so then add to a list of URIs for XooML fragments to be rendered.
6. For URIs in this list, Planz repeats steps 1 thru 4.

In this manner, Planz recursively constructs a document-like outline view for a folder and its contents. This "document" corresponds, in the extent to which subfolders (and sub-subfolders, etc.) are expanded, to exactly the view that the user last constructed in Planz.

The XooML schema…

1. Is *modular* – one "fragment" of metadata per grouping item.
2. Abstracts *common attributes* and a basic *tool-independent structure* of links.
3. Allows for additional "tool-accommodating" attributes that may also influence a user's perception of structure.
4. Provides for a simple use of namespace declarations so that any number of tools can work with the same structures without risk of confusion or collision between the attributes used by different tools.

These properties of XooML are summarized in Table 1. In

Table 2, we then see how the same grouping item can appear and be used in distinct ways through tool-(namespace-)specific attributes.

Table 1. The XooML approach to structure.

Nature of structure	**Characteristic of XooML**	**Constructs of XooML**
Explicit	Tool-independent	Fragment (node) and its zero or more associations (links)
Implicit or latent	Tool-accommodating	Namespace-specific sub-elements (uniquely identified by the "xmlns" attribute).

Table 2. Through the addition of attributes in namespace sub-elements, the same grouping item can appear as a heading in Planz, a node in a mind-map in FreeMind and a note on a page in a OneNote clone.

	Fragment	Association	Comment
Common[8]	*schemaVersion* *itemDescribed* …	***ID*** ***displayName*** *assoc.XooMLFragment* …	All "XooML-speaking" tools are expected to make proper use of common attributes. Tools will read and modify these attributes as needed.

[8] Other common attributes include *createdBy (URI), createdOn (time/date), lastModifiedBy* and *lastModifiedOn.*

Planz	*showAssocsMarkedDone* *showAssocsMarkedDefer, …*	*isVisible* *isHeading* *isCollapsed, …*	Planz uses additional attributes to determine, for example, whether an association should appear and, if so, as a heading or note and if, as a heading, whether expanded or collapsed.
FreeMind		radius polarAngle, …	An open-source mind-mapping application FreeMInd is modified to work with XooML. Polar coordinate attributes for each association determine the relative positioning of display links in a "mind map"
OneNote'		XOffsetForNote YOffsetForNote TextOfNote	A OneNote clone might position these same associations as notes on a "page". Namespace-specific attributes determine the positioning of each note and the text to display.

Some variations in the use of XooML

1. **Support a metadata standard**. A collection of applications might all work with the information in a namespace sub-element. For example, applications self-described as supporting Dublin Core might each work with sub-elements (at both the fragment and association levels) accordingly identified (e.g., xmlns:dc="http://purl.org/dc/elements/1.1/"). Other tools might work with iCalendar namespace bundles (e.g. with `xmlns="urn:ietf:params:xml:ns:icalendar-2.0"`).
2. **Make bibliographic references, tasks and to-dos**. As an extension to variation #1, namespace sub-elements needn't be restricted just to "surface" information for display (e.g., position, color, shape, etc.). Sub-elements can contain information needed for an association or a fragment to work (appear and behave) as a task, an appointment, a bibliographic reference, etc. For example, an iCalendar attribute bundle could provide the necessary data for a fragment or an association to behave as an event (IC:vevent) or a "to-do" (IC:vtodo).[9]
3. **Use associations or whole fragments?** As #2 suggests, the namespace bundle needed to make an association work in a special way (e.g., as a task, an appointment, a reference, etc.) could just as easily be placed at the level of the fragment as a whole. Fragment or association? The answer depends upon whether we wish for the "thing" involved to

[9] XooML2's allowance for namespace elements to themselves take sub-elements makes support for standards (and associated XML schema) much easier. But then tools need to exercise care to insure that these sub-elements don't end up containing tool-independent structure.

behave as a grouping item in its own right – e.g., as capable of linking to other items and, in turn, capable of being linked to.[10]

4. **Support RD**F. Namespace bundles can also be used in support of RDF (e.g. xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"). A fragment-level bundle might, for example, include the data of an rdf:Description element.[11] Alternatively, each of a fragment's associations might be further qualified through an assignment of the rdf:predicate (where the object is the item pointed to by the association).[12] More generally, the grouping item and its metadata can represent a subject of SPO propositions. The grouping item, through its outgoing links, then groups together the "things we know about" this subject. We are again reminded of the value of the grouping item as a provider of context. Some relationships among associations (facts, propositions) only emerge within the context of the grouping item.
5. **Represent a hypergraph**. In example #4 and with reference to graph theory, an association can be regarded as an edge with 3 vertices: one each for the subject, predicate and object of proposition. More generally, the provision for namespace elements in which attributes can "link" (via URI addresses pointing to other items) means that the association as an node (edge) can have any number of links (vertices) i.e. namespace elements allow us to represent a hypergraph <ref>.

[10] . In this respect, folders as a grouping item are especially plastic. Given the right sub-element information within its associated XooML.xml file, a folder can be made to appear in many different ways.

[11] But then a reasonable restriction is that URI in the value of the *rdf:about* attribute should be the same as the URI for the grouping item that is described by the XooML fragment.

[12] i.e. rdf:object is not used or, if used, is enforced to have the same value as the item linked to by the association. Likewise, rdf:subject or rdf:about are not used or are restricted to have URI value equal to the URI for the fragment's grouping item.

The actual history of XooML's use and the events leading up to XooML.

Work leading up to XooML is roughly as follows:

2005-2006. The "Universal Labeler"[13] was designed to turn file system folders in to a "personal unifying taxonomy" (PUT) able to tag and organize not only files but also web pages, email messages and informal note. *One organization for many different forms of information.* The Universal Labeler had only an alpha existence as a "hands-on" proof-of-concept demonstration. [14]

2007-20010. ***The Personal Project Planner* => (later changing in name to) *Planz*[15] is developed** as a full-featured prototype according to a guiding principle that people should be able to use an existing organization – the file system hierarchy – as a PUT to organize all digital forms of information. *One organization for many different forms of information.* Planz is reasonably robust and full-featured and can be used (is being used) by people on an ongoing basis.

20011-present. *XooML* for a vision: Many tools, many stores, one unifying structure[16]. Why only one tool (Planz) as an overlay to only one organization (the file folder hierarchy)? Why not many tools, used interchangeably, on the same organization? And then why not a unifying structure that extends across various digital stores to encompass all of our digital information and not just the information in our files and folders? Even as work on Planz continued, XooML was developed as a representation of structure. XooML is used today in Planz and a QuickCapture prototype that are available for free download[17]. XooML has also been used in a proof-of-concept "alpha" wrapping of the FreeMind open-source mind-mapping tool. More recently, XooML has been used combination with Drobox to support multi-person, multi-platform work on the same structure. The structure appears as a 1.A document-like outline in Planz or a file folder hierarchy as viewed form a Windows machine and then also 2. As piles on "bulletin boards" via *MindCloud*, an affinity diagraming tool that runs on an iPad <ref>.

[13] (W. Jones, Bruce, Foxley, & Munat, 2006; W. Jones, Munat, & Bruce, 2005)

[14] In a half-hour session, participants were led through a use of the prototype but only along certain paths.

[15] (W. Jones, Klasnja, Civan, & Adcock, 2008; William Jones, Hou, Sethanandha, & Bi, 2009; William Jones, Hou, Sethanandha, Bi, & Gemmell, 2010).

[16] (W. Jones, Anderson, & Whittaker, in Press; William Jones, 2011; William Jones & Anderson, 2011).

[17] http://kftf.ischool.washington.edu/planner_index.htm.

IV. What is XooML2 (and why)?

Work on XooML – both XooML2 and earlier versions -- is motivated by the following observations:

1. **Our information is scattered**, ever more so, by the by the very devices and applications we use to manage this information.
2. **Processes of export/import or synchronization aren't enough**. These processes are cumbersome and often error-prone. Most notably, export/import procedures frequently fail to transfer the structures we use to work with and make sense of our information.
3. **There is no such thing as a "super-tool" that might "save" our information** from a current state of growing fragmentation. Any tool that purports to do this simply adds to the problem.
4. We **look instead for representations of our information structure that are tool-independent and also tool-accommodating**. We look for representations that make it possible for us to select among a variety of tools as we tend our growing gardens of information.
5. A tool-independent but tool-accommodating representation of structure can be realized through a **focus on the *grouping item* and its underlying "noodle" of explicit structure** (a node + outgoing links). The grouping item is a basic unit of structure.
6. All of our tools support the grouping item in one form or another. If the simple underlying "noodle" structure of these items can be read and represented in a tool-independent way (e.g., in XML) then **we begin to unlock our information structures from supporting tools**. We can consider, critique, compare and copy our structures.
7. As a step further, if our representation of structure is tool-accommodating and **if it is possible to write back changes to this structure**, then we can begin to separate, in our selection of tools, between support for surface features of information interface and manipulation and **store**-level features such as safety, security, synchronization and sharing. Our information – content and structure – becomes a middle layer connecting storage and surface layers.

A proof-of-concept for XooML exists in its use (version 0.41)[18] across several prototype tools of information management: Planz, QuickCapture and FreeMindX <ref>[19]. These tools and also the Windows Explorer can be used interchangeably with the same information structure which, in turn, is persisted as a folder hierarchy in the Windows file system. More recently, a newer version of XooML has formed the basis for an interchangeable use of Planz from a Windows machine and MindCloud from an iPad as applied to the same information – stored in file folders and shared via Dropbox<ref>.

[18] XooML was released in beta as version 0.41 (http://kftf.ischool.washington.edu/XMLschema/0.41/XooML.xsd).

[19] For a download of Planz and QuickCapture visit: http://kftf.ischool.washington.edu/planner_index.htm.

XooML2 (draft)

XooML2 incorporates the practical lessons learned from the use of XooML in these prototypes. Specifically, attributes are removed that were never used. Also restrictions in the composition of namespace sub-elements (more about this below) are removed in order to be more "tool-accommodating" and also in order to support the schemas of metadata standards such as Dublin Core.

In addition, XooML2 is designed to support:

- **Group use.** XooML2 provides explicit mechanisms for the read and write of shared XooML fragments as done in the same timeframe but through different tools used by different people.
- **Multiple stores.** XooML2 moves beyond specific stores as supported by Windows on the desktop or through Dropbox on the Web to a more general treatment of the stores supported by different applications (and accessible through APIs). New attributes make provision for utilities that provide a consistent "face" to back-end interactions that vary depending upon the store and its supporting application (e.g., Drobox, Facebook, Evernote, etc.). General provision is made not only for variations in the store of grouping items but also for variations in the way corresponding XML fragments of metadata are stored.
- **A snapshot log[20] for versioning and rollback of metadata fragments**. Provision is made (in an optional "Extras" namespace sub-element of a fragment) for specification of a location and utilities for creation of context-stamped (time, place, person, etc.) "snapshots" of the metadata fragment. These can then provide the basis for rollback, recovery of view settings and, from an earlier rollback, a "play forward" to see a kind of time-lapse animation of the changes in grouping items (as displayed in the "document" constructed by a given XooML-speaking tool).
- **An item event log for interactions with the grouping item**. Separate provision is made (also in the optional "Extras" namespace sub-element) for auxiliary XML structures and associated utilities to log interactions with the grouping item (e.g., folder, tag, "project", etc.) described by the metadata of a XooML fragment. An "i.e log" stores an interaction

[20] Provisions for versioning and rollback focus on the preservation of "snapshots" of a XooML fragment. Using these snapshots, it is possible to rollback to earlier versions of a fragment or assemblage of fragments (i.e. assembled into a document for viewing through a XooML-speaking tool). However, this view is generally read-only since there is no corresponding ability to roll-back the state of a grouping item to its earlier version.

Focus in the i.e.log on the other hand is on the grouping item. What is the history of interactions with this item? If the log is complete and there is undo support in the store(s) for grouping items, then the log provides the basis for a rollback in the state of a grouping item to an earlier point in time. If a snapshot of the associated XooML fragments is also taken at this earlier point in time, then it is possible to have a complete rollback – of grouping items and the XooML fragments describing these grouping items.

with the item as an event (an item event) in two parts: the **intent** (e.g., the attempt to open, move, re-name, create, remove, etc.) and the **effect** (outcome): i.e., did the attempt fail or succeed?

Changes in the schemas for XooML2 fall into two broad categories:

1. **XooML "essential".** The XooML2-essentials.xsd defines a XooML fragment that is as simple as "can be".
    a. Many attributes are eliminated or pushed into to "extra" namespace elements for optional use. The attributes that remain are named to be more descriptive of their use.
    b. Restrictions on the namespace sub-elements in earlier versions of the XooML schema precluded support for standards such as Dublin Core. XooML2 relaxes these restrictions so that a sub-element, as used by a specific tool or a class of tools (e.g., tools supporting Dublin Core or tools supporting iCalendar) can contain the necessary the XML structures to support common metadata standards.
    c. On the other hand, XooML2 adds attributes pointing to utilities to promote more consistency in the way grouping items are read and manipulated and in the way XooML fragments are written to storage. Also, there is provision for a shared use of a synchronization utility to insure that the metadata of a fragment remains consistent with the "reality" of the grouping item it is meant to describe.
2. **XooML "extras".** XooML2 additionally provides for optional inclusion of namespace elements – both at the level of a fragment and for each of its associations – that contain additional attributes in support of item event logging, versioning & rollback. The i.e.log and the snapshot log for a given XooML fragment (describing a given grouping item) are stored separately from the fragment according to separate XML schemas.

Subsections under the this "XooML2" section are as then follows:

1. XooML essentials
2. XooML extras
3. Step by step use of XooML, in the general case.
4. Specific examples of XooML use. We consider its use with a file system + Dropbox and then with Evernote.
5. FAQs (fairly apparent questions)
6. Issues & next steps.

Some caveats and disclaimers

- The design of XooML2 is recently complete and no tools yet use it. XooML2 is designed to address imperfections in earlier versions of XooML. But doubtless in the course of its use, new imperfections will become apparent (hence the need for schema versioning).
- One possible change on the horizon: A conversation of all common attributes (fragment- and association-level) to elements for greater flexibility in representation.
- Many attributes are removed from the basic ("essential") schema for a XooML fragment. Most of those that remain have been renamed. New names were selected to more clearly reflect the model for XooML's use.

XooML2 essentials

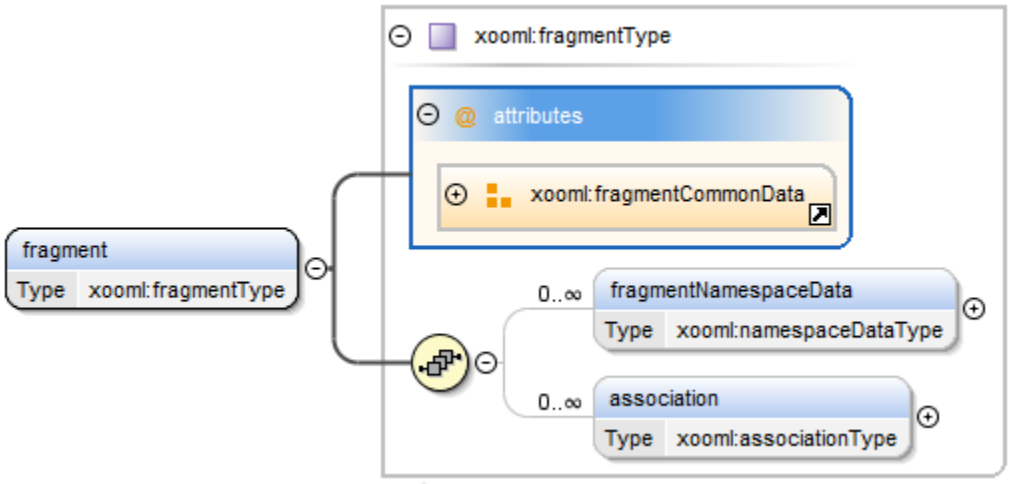

Figure 5. A fragment is a bundling of "common" (tool-independent) attributes followed by zero or more bundles of "tool-accommodating" attributes (fragmentNamespaceData sub-elements) followed by zero or more associations.

The schema for a XooML fragment (see Figure 5) is much the same as before. A fragment is an XML element containing:

1. Attributes describing "*fragmentCommonData*" that is used by all tools making use of the fragment. Most notably, there is an attribute named "itemDescribed" taking, as a value, a URI pointing to the item described by the fragment.
2. Zero or more *fragmentNamespaceData* sub-elements bundle data that applies to the fragment overall but is specific to the fragment's use by a tool or set of tools.
3. Zero or more associations just as before.

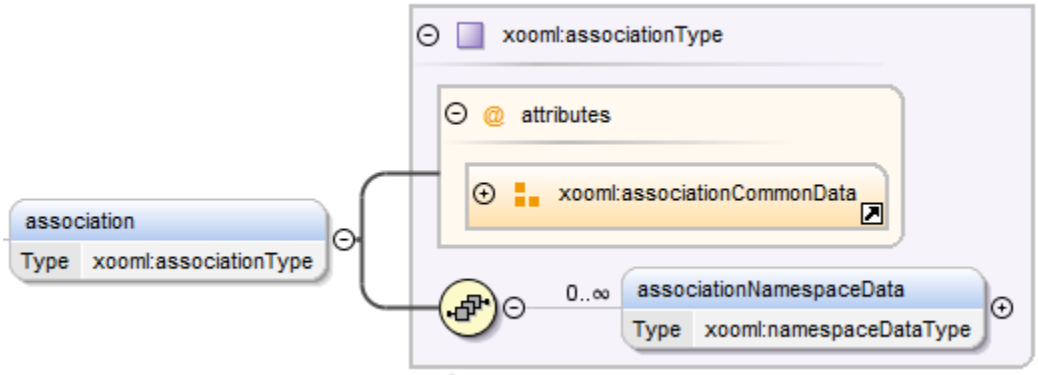

Figure 6. Each association is also is a bundling of "common" (tool-independent) attributes followed by zero or more bundles of "tool-accommodating" attributes (associationNamespaceData sub-elements).

Each association in turn (Figure 6) is an element containing:

1. Attributes describing "*associationCommonData*" that is used by all tools making use of the association. Most notably, there is an attribute named "associatedXooMLFragment" which optionally can take as a value, a URI pointing another XooML fragment. This fragment is generally metadata for another grouping item pointed to by the "itemDescribed" grouping item represented by the current fragment. For example, if the current fragment represents *Folder A* and *Folder A* contains *Folder B,* then the XooML fragment that describes *Folder A* would have an association to another XooML fragment that describes *Folder B.*
2. Zero or more *associationNamespaceData* sub-elements bundling data that is specific to the association's use by a tool or set of tools. Planz, for example, provides for an attribute called "isCollapsed" that is specific to its "Planz" namespace sub-element for each association. The value of this attribute (TRUE or FALSE) determines whether the association should be "expanded" in the outline view of Planz to form a display based upon the *associatedXooMLFragment.*

These essentials of the fragment and its associations under XooML2 realize the following desirable improvements:

1. **Greater simplification and clarity**. Unused, unnecessary attributes are gone. The names of types and remaining attributes have then been changed to be more descriptive. For example, the "relatedItem" attribute has been renamed, *itemDescribed.* Instead of "toolAttributesType" there is now the *namespaceDataType* to be used for bundles of "tool-accommodating" data for both the fragment overall and for each of its associations. The change to "namespace" allows for the possibility that several tools

may work in the same namespace (for example, that for Dublin Core or for iCalendar). The change from "Attributes" to "Data" is in recognition of the removal a restriction that namespace bundles can only contain attributes and values. (More about this next.)

2. **Greater flexibility**. As a general rule of "good style", a namespace sub-element should not include tool-independent or "noodle" (node + outgoing links) information. Good style was enforced in the XooML 0.41 schema through a restriction that a namespace sub-element could contain only attributes. However, this restriction did not allow for support of metadata standards such as Dublin Core where, though metadata is essentially a set of attributes (e.g., "creator"), these are represented as elements in the namespace schema and can then, themselves, take attributes (e.g., to establish provenance). The restriction has been removed. The expectation of good style remains.
3. **Greater consistency and flexibility in the manipulation of grouping items and in the storage and synchronization of XooML fragments**. Provision is made in XooML2 for common, fragment-level attributes that point (via URIs) to "utilities" to provide consistent handling of three essential activities: a. reading/manipulating the grouping item (itemDescribed), b. synchronization to identify the changes that need to be made to the XooML fragment in order for its metadata to agree with the structure of the grouping item and c. writing of modified fragment metadata back to storage.

Fragment common data attributes, including those for the utilities, are now described in greater detail. This is followed by a more in-depth description of the association common data attributes.

Attributes for *fragmentCommonData*

schemaVersion (required). The current version of the XooML schema in effect for the fragment.

itemDescribed (optional). A URI pointing to item described by the metadata of a fragment. A URI might point to just about anything that can be interpreted as a grouping item. For example: a conventional file system folder, a "tag" as supported by any off several applications, or an application-specific construct such as a "notebook" in Evernote, a section tab in OneNote or an "album" in Facebook. Even the term of an index built through a indexing utility such as Lucene can be interpreted as a grouping item (i.e., it is addressable and, in turn, directly addresses documents in which it was found as content).[21]

[21] An ordinary Web page can be considered as a grouping item with structure in the form of hyperlinks.

Why is itemDescribed optional? A XooML fragment is metadata for an item. But this item may not exist as a thing to be returned through the resolution of a URI. The item might be a mental list of "things to do over the weekend".

utilityToWorkWithitemDescribed (optional). Different utilities are needed for different item stores (as supported through different interfaces). To work with (list, open, move, delete, create, re-name, etc.) a Windows file system folder, for example, we need a utility that can "speak" Win32. To work with the grouping items of Evernote, we need a utility that works through the Evernote Cloud API. To work with the grouping items supported through Facebook, we need a utility that works with the Graph API. And so on… Utilities, though necessarily diverse on the backend, might all themselves support a common interface for use across tools – for example, the File API (http://www.w3.org/TR/FileAPI/).

utilityToSynchronizeFromitemDescribedToXooMLFragment (optional). This utility compares the data returned for the **itemDescribed** (through use of **utilityToWorkWithitemDescribed)** with corresponding metadata of the fragment. In particular, the synchronization utility compares the listing of contents for a grouping item with the listing of associations for a fragment. The utility outputs a listing of changes (e.g., as xPath expressions) needed to bring the fragment into alignment with the "reality" of the itemDescribed. (In cases of conflict, the itemDescribed always wins). These changes are then saved to the store for the XooML fragment (see next attribute).

Note that this attribute, by specifying a synchronization utility, provides a more general replacement of the "levelOfSynchronization" attribute in earlier versions of XooML. There are many models of synchronization and these do not neatly fit on an ordinal scale.[22]

utilityToWriteXooMLFragment (optional). Just as the itemDescribed may come from many different stores, so too the XooML fragment that describes this grouping item may be saved in different ways. When file system folders are the grouping items to be described, it has worked well (e.g., in Planz and QuickCapture) to save a folder's XooML fragment simply as a file ("xooml.xml") within the folder. Even as the folder is renamed or moved, the fragment remains. But for grouping items from other stores supported by other applications, such a solution won't always work. In some cases, for example, the user (and the tool he/she is using) may not have write access to the store of a grouping item. Other solutions include the use of a database (e.g., MySQL). In general, it is possible that the store for XooML fragments is managed very differently than is the store(s) for grouping items.

[22] For example, Planz supports the persistence of a simple, plain-text note that is not linked to an information item and has no file system counterpart (i.e., in the contents of a folder as the itemDescribed). The synchronization utility "overlooks" this lack of correspondence.

GUIDgeneratedOnLastWrite (required)**.** This is re-set for a XooML fragment with each successful write. A XooML-speaking tool reads this attribute along with the rest of the XooML fragment. Immediately prior to any write attempt, the tool reads the fragment again to verify that the GUID value for the attribute is the same. If so, then it "knows" that the XooML fragment it last read is still the last one to be written. It can proceed with a write. On the other hand, if the GUIDs don't match then a newer version of the fragment has been saved (by another user through another tool). The tool should read and work with this newer version of the fragment. Any eventual changes made and written out must be made through this newer version. (See the sub-section below "Step by step use of XooML…")

Note*: The use of utilities attributes is optional but strongly recommended. These specify common utilities for consistency in the handling of fragments, items and the synchronization between these. If an utility attribute has a valid non-null value (pointing to an operational utility) then all XooML-speaking tools that work with the fragment are expected to use this utility.*

Attributes for *associationCommonData*

ID (required). Each association element is uniquely identified through a GUID.

displayName (optional). Each association optionally has a string of text that can be used to represent and describe the association in displays. (Associations can also optionally have a display icon through the "Extras" namespace sub-element. See below.) Though displayName is optional, its use is strongly recommended. If displayName is non-null, XooML-speaking tools are expected to find a way to show this name in their renderings of an association.

associatedXooMLFragment (optional). An association can optionally point, though this attribute's URI value, to another XooML fragment describing another (grouping) item.

XooML2 Extras

In this section, we consider "extra" attributes that, though not essential to the use XooML, are very useful. "Extra" attributes are packaged into namespace elements and can be used both for the fragment as a whole and each of its associations. Namespace elements at each level are identified by the namespace declaration: xmlns:xooml-extras=http://kftf.ischool.washington.edu/xmlns/xooml-extras.

Note: A *context* type is defined and instantiated in several places within the "Extras" schemas and also the schemas for item event and snapshot logging. *Context* encapsulates attributes with the following names: *who* (a URI identifying the user associated with an event), *when,*

where (GPS coordinates) and *withWhatTool* (a URI specifying which tool was in use).[23] In descriptions that follow, we will simply refer to these as the "context attributes".

We'll consider first the Extras at the association level and then the Extras at the fragment level. Then we'll consider the structure of the two logs: the item-event log and the snapshot log.

Association Extras

firstCreated is an element that includes the context attributes and, as its name implies, records the circumstances of an association's creation. Note that there is no complementary provision for a "lastModified" since this information can be recorded in the logs.

itemDescribedbyAssociatedXooMLFragment is an element with attributes to cache useful data concerning the information item pointed to by an association. This data is derived from other sources and care should be taken to insure that the data agrees with the data from these sources. The following attributes are present:

- itemAddress – a URI that should agree with the URI of the itemDescribed attribute in the fragment pointed to by the associatedXooMLFragment of the association.
- itemApplicationToOpen – a URI pointing a default application to be used to open the item.
- itemIcon. To be used in the display of the association and its associated item.[24]
- itemPop-upText – text to display upon a hover over the icon for the associated item.

Fragment Extras

Extras at the fragment level include:

firstCreated and **lastWritten** elements. Both include the context attributes. lastWritten context attributes are given new values with each successful use of the utilityToWriteXooMLFragment. firstCreated context attributes are given values upon initial creation of the fragment (which may not necessarily coincide with the creation of the itemDescribed).[25]

displayName -- a text string to be used in representations of the itemDescribed of the fragment (e.g., the title of the display window for the fragment and its itemDescribed).

itemEventLog – an element packaging the following attributes:

[23] The "what" of an event is not recorded.

[24] There may eventually need to be several icon attributes to accommodate the needs of different tools in need of differently-sized icons.

[25] In Planz, for example, XooML fragments are created "lazily" i.e. only as needed. If there is a need to display and work with the contents of folder and this folder does not yet have a fragment (in a xooml.xml file) then the fragment is created.

- itemEventLogLocation – a URI pointing to the base file or record of the log.
- itemEventLogWriteUtility – a URI pointing to the utility that writes out a new event.
- itemEventLogReadUtility – URI pointing to the utility that reads the item event log.

The structure of the item-event log is described in the next sub-section. Here we note that the writing of a new event to the log may be a simpler operation than reading the log. New item events can simply be appended to the log or even written in a separate file/record to be appended to the source log later. Even if item events happen to be written out of order with respect to their occurrence, a temporal ordering can be established later: Item events each carry the context attributes.

Reading from the log is a very different operation. Parameters for the read utility are not for us to specify here but even basic operations like "list all events of the past week" require that the utility be able to read the entire log – merging, as needed, log fragments.

snapshotLog – an element packaging the following attributes:

- snapshotLogLocation – a URI pointing to the base file or record of the log.
- snapshotLogWriteUtility – a URI pointing to the utility that writes out a new event.
- snapshotLogRollbackUtility – URI pointing to the utility that reads the item event log.

As with the item event log, writing to the snapshot log can be a simple append with a new ancillary file/record started as needed should the log itself be locked. Snapshots are stamped for time by their context attributes and so temporal ordering can be recovered later as needed. As will become clearer in sub-section describing the schema for the snapshot log, snapshots can be represented by a simple copy of the XooML fragment. Later, to save space, this copy might be replaced by a "delta" offset computed –directly or indirectly – from the XooML fragment in its current state.

In contrast to the write utility and in line with the read utility for the item event log, the operations of the snapshotLogRollbackUtility are more intricate. Again, parameters of this utility are not for us to specify here but certainly most operations require a read of the entire log. The rollback to the fragment at a previous point in time may require the application of a succession of deltas staring with the fragment in its current state (e.g., fragment(n)) and recovering a succession of earlier states (fragment(n-1), fragment (n-2), etc.) until the state at or "near" the desired time point is reached.

Note that logs relating to a XooML fragment and its itemDescribed are kept separate from the fragment for the simple reason that logs, unless trimmed, grow ever larger and would impose an ever greater time penalty if stored in the fragment itself.[26]

The Item-Event Log

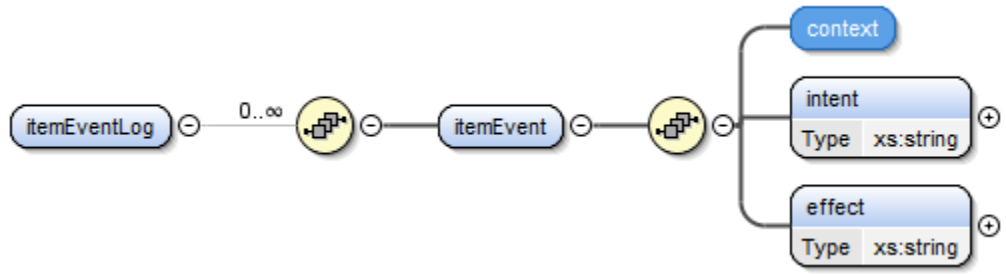

Figure 7. A depiction of the schema for the item-event log.

The schema for the item-event log is depicted in Figure 7. The log is written to (using itemEventLogWriteUtility) as a by-product of a tool's invocation of the utilityToWorkWithitemDescribed[27]The log is composed of zero or more itemEvents. Each itemEvent includes a context (who?, when?, where?, with what tool?). The "what" of the item event is in two parts: the "intent" is a string encoding a File API[28] request sent through the utilityToWorkWithitemDescribed via API to the application for the itemDescribed (e.g., Win32 if the itemDescribed is a Windows file folder). The "effect" is a string encoding – also in the language of the FileAPI -- the outcome of this request (including its success or failure).

The Snapshot Log

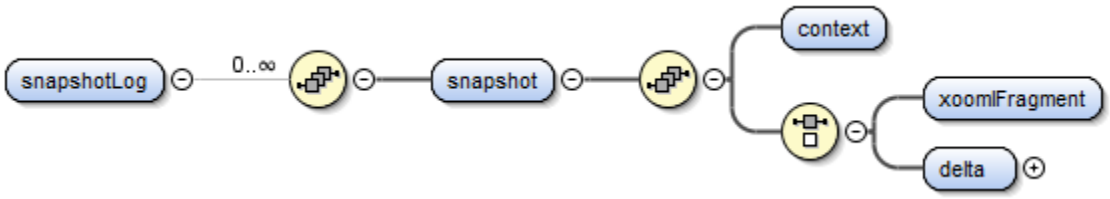

Figure 8. A depiction of the schema for the snapshot log.

[26] On the other hand, the size of a fragment does not necessarily get larger with the simple passage of time. It should grow larger (or smaller) in accordance with the size of its itemDescribed and also grows larger according to the number of tools used to work with the itemDescribed.

[27] Is every interaction with an itemDescribed recorded or only certain "important" interactions (such as a move or modification)? For now, this is policy determined by the itemEventLogWriteUtility. There may be reason later to control log fidelity through a setting on the log itself.

[28] http://www.w3.org/TR/FileAPI/.

The schema for the snapshot log is depicted in Figure 8. The log consists of zero or more snapshots. A snapshot, like an item event, is "stamped" with the context attributes. A snapshot is written via the snapshotLogWriteUtility called from the utilityToWriteXooMLFragment. As with the item-event log, the determination over when and how often to take a snapshot is currently determined by snapshotLogWriteUtility. [29]

The structure of an snapshot includes a choice between a simple copy of the XooML fragment and the use of a delta representing the changes to be applied to a more recent fragment(n) in order to recover the state of a fragment(n-1) representing the fragment at the time that the snapshot is taken. Deltas can be computed in any of a number of ways.[30] It makes no sense to compute a delta at the time that a snapshot is taken. (Since the copy of the fragment is identical to the fragment, the delta would be null). Deltas can be computed later "lazily" as needed in order to reduce the size of the log. Or, as the snapshotLogWriteUtility creates a new snapshot, it might replace the XooML fragment copy for the previous snapshot with a delta representing operations to be applied to the current fragment(n) to recover the previous fragment(n-1).

Some variations in the use of the logs.

- The snapshot log might be inspected to recover view settings for a given user even if these have been overridden in the current fragment by another user's preferences.
- The item-event log + the snapshot log + plus an ability to "undo" as supported by the store of the itemDescribed = an ability to rollback not only the state of the fragment (or a larger "document") but also to rollback to an earlier reality with respect to the actual information items.

Step by step use of XooML, in the general case.

In this sub-section, we consider a step-by-step use of XooML2. As a reminder, fragment-level attributes specify the following:

- **utilityToWriteXooMLFragment** (shortened to "Write Fragment"), a URI pointing to a utility for writing (and also reading) a XooML fragment (as referenced through a URI). This

[29] A revision of the snapshot schema may make provision for attributes in the snapshot log to specify, for example, that a new snapshot is to be recorded with every successful write of the XooML fragment.

[30] There are two general solutions to the computation of the delta. Solution #1 is treat the fragment as a plain text string. There are then a number of utilities for computing the difference between two strings (e.g." diff"). Solution #2 would leverage the structure of XML. Deltas could be expressed as a sequence of XPath expressions. For more information, see:

http://pages.cs.wisc.edu/~yuanwang/papers/xdiff.pdf, http://www.stylusstudio.com/xml_differencing.html, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1318581&userType=inst.

utility makes use of and is specific to the API supported by a storage management facility like Dropbox or Windows. The utility takes as input a string for the XooML fragment together with a URI pointing to the storage location for the fragment. The utility outputs either "success" or, on failure, a string representing the fragment as just retrieved. The utility can optionally call the snapshotLogWriteUtility to make a save a version of the fragment (e.g., as it was in storage before the write).

- **utilityToWorkWithitemDescribed** (shortened to "Work w Item"), a URI pointing to a utility that uses the API of a storage management facility in order to read and make changes to the itemDescribed by a XooML fragment. Operations supported are more fine-tuned than for Write Fragment. To use the example of a folder as a itemDescribed, the utility needs to support an ability to "List" the folder's contents and to add/delete from these contents (e.g., add/delete a subfolder, shortcut or other file). The utility takes as input a File API expression of intent (i.e. what is to be done with the item described). The utility returns the "effect" of this request (also a File API expression). The utility can, optionally, call the itemEventLogWriteUtility in order to log the interaction with the itemDescribed (as intent and effect).
- **utilityToSynchronizeFromitemDescribedToXooMLFragment** (shortened to "Synchronize"), a URI pointing to a utility that compares a listing of contents for a itemDescribed (as delivered by Work w Item) with comparable content (associations) in a XooML fragment. The utility takes as input a string representing the XooML fragment. The utility returns a XooML string modified to bring its content in line with the content of the itemDescribed or null, if no changes were necessary. The synchronization utility speaks the File API[31] in is interactions with Work w Item.
- **GUIDgeneratedOnLastWrite.** This attribute is re-set for a XooML fragment with each write attempt. A XooML-speaking tool reads this attribute along with the rest of the XooML fragment. Immediately prior to any write attempt, the tool reads the fragment again to verify that the GUID value for the attribute is the same

[31] http://www.w3.org/TR/FileAPI/.

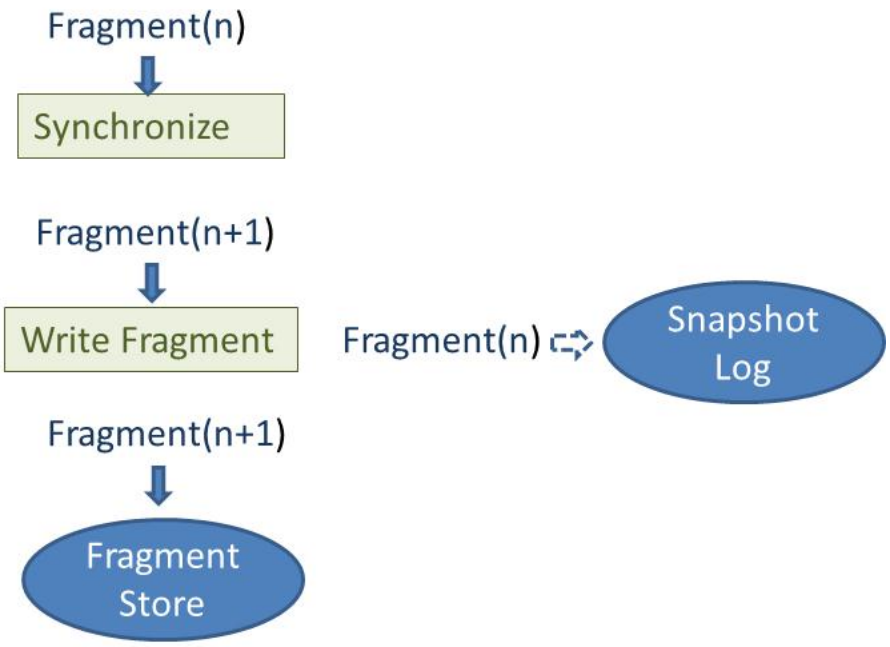

Figure 9. Preliminary steps in the creation of the view for a fragment (Case A). Keeping a snapshot is optional.

Consider then, the situation where Tool T, as used by person P, reads a set of XooML fragments in order to construct a "document" view (where, depending upon tool T, the view may appear as a Planz-like outline or as stacks in a Mindcloud bulletin board or as something else like a mindmap). Call this case A (see Figure 9).

Any given tool T will work with a StackOfURIsToXooMLFragmentsToRender. Initially the stack will have a single URI pointing to a single XooML fragment. The sequence of operations in case A is then as follows:

1. Pop the StackOfURIsToXooMLFragmentsToRender. If empty, stop.
2. Otherwise, resolve the URI to a XooML fragment F. Note the value of the GUIDgeneratedOnLastWrite. Call this G(read).
3. Use Synchronize (which in turn works with utilityToWorkWithitemDescribed) to determine a list of changes that need to be made to the fragment F in order to bring its data (e.g., display name, associations) into agreement with those of the itemDescribed.
4. Make these changes to the tool-specific structure for fragment F.
5. **Critical step:** Using Write Fragment with G(read) as an argument attempt to write a serialization of the fragment structure back to the XooML fragment store. **The utility will fail (and return an error, raise an exception)** if the "current" GUIDgeneratedOnLastWrite – call this G(write) -- does not equal G(read). If not equal, then go back to step 2. If equal (success!) then continue to step 6.

6. For each association in Fragment F, inspect tool-specific attributes to determine whether its associatedXooMLFragment should also be rendered in the current view (document)l. For Planz this means looking to see if the isCollapsed attribute is FALSE. If so, then add the URI value of its associatedXooMLFragment to the StackOfURIsToXooMLFragmentsToRender.
7. Return to step #1.

Issue: How often will failure occur at step #5? And can we ever face a situation of deadlock? In worst case two tools may compete with one another to see which tool gets to commit the results of synchronization. If tool T and tool U both attempt a synchronization at the same time, one – say tool T, will be first. Then the other tool will fail at step 5 and will then go back to step 2. Only this time the synchronization (step #3) should produce an empty list of changes that need to be made.[32]

Now consider variations of a case B where tool T has already rendered a set of fragments for the current view which is not being used by person P. What happens as these fragments are also being viewed and possibly modified by another person Q through tool U? We'll consider with Planz as tool T. In one variation of case B, Person P sees a folder F in Planz as a heading H which is collapsed. The click to expand is really simply a special case of A – the initial generation of a view. In this variation, the URI for the associatedXooML fragment is pushed onto the StackOfURIsToXooMLFragmentsToRender and then steps #1 through #7 above are followed as before.

[32] Note that the locking of the record or file for containing the XooML fragment is needed only for as long as the write process needs to complete (and could be set by the storage facility to timeout).

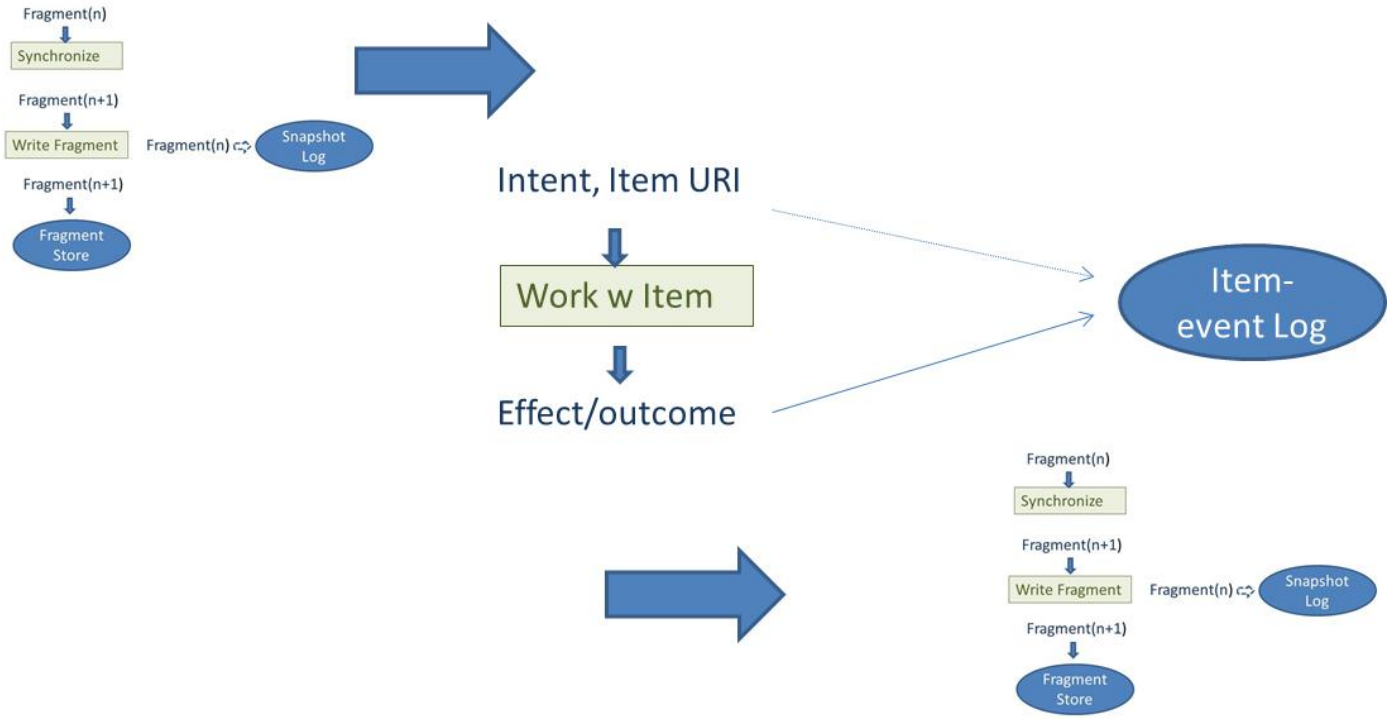

Figure 10. Steps involved (Case B) when the user wishes to work with the contents of an item (e.g., open, create, or delete a child item). Note that Synchronize and a Write Fragment (as needed) happen both before and after the Work w Item.

In a more interesting variation of Case B, the heading H for folder F is already expanded and associations from the synchronized fragment for folder F are in view. In Planz these appear as notes (for files) and subheadings (for subfolders). The region rendered for a fragment is called a Plan. Person P may wish to do something with this Plan and its associations. For example, person P may wish to open the item pointed to by an association into a separate application. Person P may wish to delete an association and the item to which it points. But first, it should be established that the region as rendered (the Plan) is still current. Verification happens as follows (see also Figure 10):

8. Upon a shift of focus (e.g., as communicated by click or hover) to the region defined by the XooML fragment, the sequence of steps under Case A above are followed first to insure that the tool's copy of the fragment is still current (i.e., that it has not been superseded by a more recent write of the fragment and that it is still consistent with the itemDescribed). If the fragment is must be updated then the region in view is re-rendered.
9. The requested action (open, create, rename, delete, etc.) is handled through the Work w Item. The request is optionally logged as an *intent* in the fragment's associated item-event log. The *effect* (outcome) of the request, whether failure or success, is also

logged[33]. (See description of logs in the "Extras" section above.) On failure, raise an error for the person P to see (e.g., "Cannot delete, file still in use…") and stop. With success continue to next step.

10. the sequence of steps under Case A above are then followed a second time to determine what changes if any must be made to the fragment as a result of the operations to the content of the itemDescribed.
    Note: The synchronization of step is a good idea even in cases where the requested operation does not change the itemDescribed. Why? Because the itemDescribed may (at any time) have been changed by its "owning application" (which knows nothing of XooML or XooML-speaking tools).
    Note: step #10 is the same as step 3. Concluding steps #11 and #12 are the same as steps #4 and #5 above.

Store-specific examples of use

Consider, for example, "albums" as supported in Facebook and programmatically accessed via the Graph API (http://developers.facebook.com/docs/reference/api/album/). The Graph API can be used to view and modify the contents of an album. These contents are of 3 different kinds:

- **Photos**. The photos contained in this album.
- **Likes**. The likes made on this album.
- **Comments.** The comments made on this album

In addition, each album has a cover photo.

<note essential choice (we face repeatedly) – flat list of associations with namespace attributes to distinguish between "Photos", "Likes" and "Comments" or…. Create, for each album as a grouping item, sub-groupings for "Photos", "Likes" and "Comments".>

<more examples here – for Evernote, Dropbox, OneNote (using DOM)>

FAQs

Why both provision for "snapshots" and for an item event log

Snapshots permit a complete rollback – though without the "reality" of itemDescribed.

The item event log is focused on access to and changes in the itemDescribed.

[33] Note that failure could be for any of a number of reasons having to do with the underlying storage management facility. A file cannot be deleted in the Windows file system, for example, if it is currently open in any application. Likewise, a folder cannot be deleted, moved or renamed if any of its files are current open in any application.

Why are snapshot and item-event logs saved separately from the primary XooML metadata fragment?

In word: to insure that the read of the XooML fragment is fast and does not degrade with history of use. The logs may get ever larger. Not so the metadata fragment.

Association or fragment? Which to use for special objects like "tasks", "to-dos" and "references"?

Namespace bundles of attributes can be created at both the fragment level and the level of associations. This means that either an association or a fragment can include information needed to behave like a task, to-do, bibliographic reference, etc. Use a fragment (and an associated itemDescribed) if you wish to link to other items and, in turn, wish to be able to link to the itemDescribed and its fragment.

What happens in synchronization?

This can vary with

Example: Planz creates associations that have no "reality" in the file system. These should be "synchronized" away.

How is ordering of associations represented?

Hint: If tools working with a fragment are meant to preserve different orderings of its associations, then don't use the ordering of XML associations in the stream persisted to XooML fragment storage. Instead, use namespace attributes (e.g., "before", "after").

V. Conclusions and next steps

We have a vision of information integration: Many tools working with many different forms of information possibly in many stores but one unifying information structure under our control. What does it take to realize this vision? Obviously writing a paper like this is not enough.

But we needn't wait for some new Web-based operating system. Nor do we need to entrust all of our information to some "vault" of storage. The XooML approach means leaving information where it is and using, with small modifications, existing tools. The XooML approach is extremely lightweight:

1. A basic representation of structure that is both tool-independent (a node + outgoing links) and tool-accommodating (namespace elements can contain additional data in support of a tool or a metadata standard).
2. Optional use of utilities to be shared among "XooML-speaking" tools in support for their interactions with different stores (for grouping items and for XooML fragments)

and in support of a consistent synchronization between metadata fragments and grouping items.

Utilities can be written piecemeal in accordance with our desire to use a particular store (e.g., for the Windows file system or for Dropbox) or our desire to accomplish a certain variation of synchronization. More important is the development of "XooML-speaking" tolls that might help us to understand and work with our structures as "first-class" objects. Some tools may come from the "wrapping" of existing open-source software. In this regard, it is encouraging to note that a version of FreeMind able to read and display XooML fragments took our lead developer at the time[34] only about 2 hours (and 200 lines of code).

VI. References

Bergman, O., Whittaker, S., Sanderson, M., Nachmias, R., & Ramamoorthy, A. (2010). The effect of folder structure on personal file navigation. *J. Am. Soc. Inf. Sci. Technol.*, *61*(12), 2426–2441.

Jones, W., Anderson, K., & Whittaker, S. (in Press). Representing Our Information Structures for Research and for Everyday Use. Presented at the Proceedings of the 30th International conference extended abstracts on Human factors in computing systems, ACM (2012),, Austin, TX, USA: ACM Press.

Jones, W., Bruce, H., Foxley, A., & Munat, C. (2006). Planning personal projects and organizing personal information. *69th Annual Meeting of the American Society for Information Science and Technology (ASIST 2006)* (Vol. 43). Austin, TX: American Society for Information Science & Technology.

Jones, W., Klasnja, P., Civan, A., & Adcock, M. (2008). The Personal Project Planner: Planning to Organize Personal Information. *ACM SIGCHI Conference on Human Factors in*

[34] Eric Sheng Bi, in 2010.

Computing Systems (CHI 2008) (pp. 681–684). Florence, Italy: ACM, New York, NY. doi:http://doi.acm.org/10.1145/1357054.1357162

Jones, W., Munat, C., & Bruce, H. (2005). The Universal Labeler: Plan the project and let your information follow. *68th Annual Meeting of the American Society for Information Science and Technology (ASIST 2005)* (p. TBD.). Charlotte, NC: American Society for Information Science & Technology. Retrieved from http://kftf.ischool.washington.edu/UL_ASIST05.pdf

Jones, W., Phuwanartnurak, A. J., Gill, R., & Bruce, H. (2005). Don't take my folders away! Organizing personal information to get things done. *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2005)* (Vol. 2005, pp. 1505–1508). Portland, OR: ACM Press.

Jones, William. (2011). XooML: XML in support of many tools working on a single organization of personal information. *Proceedings of the 2011 iConference* (pp. 478–488). Seattle, Washington: ACM. Retrieved from http://delivery.acm.org/10.1145/1950000/1940827/p478-jones.pdf?key1=1940827&key2=5397719921&coll=DL&dl=ACM&ip=205.175.115.100&CFID=11076833&CFTOKEN=71585935

Jones, William, & Anderson, K. M. (2011). Many Views, Many Modes, Many Tools... One Structure: Towards a Non-disruptive Integration of Personal Information. *Proceedings of the 22nd ACM conference on Hypertext and hypermedia* (pp. 113–122). Eindhoven, The Netherlands: ACM.

Jones, William, Hou, D., Sethanandha, B. D., & Bi, E. S. (2009). Planz: Writing New Stories for the Same Old File System. Presented at the PIM 2009, Vancouver, B.C. Retrieved from http://pimworkshop.org/2009/papers/jones-demo-pim2009.pdf

Jones, William, Hou, D., Sethanandha, B. D., Bi, S., & Gemmell, J. (2010). Planz to put our digital information in its place. *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems* (pp. 2803–2812). Atlanta, Georgia, USA: ACM.

Schon, D. A. (1984). *The Reflective Practitioner: How Professionals Think In Action*. Basic Books.

Teevan, J., Alvarado, C., Ackerman, M. S., & Karger, D. R. (2004). The perfect search engine Is not enough: A study of orienteering behavior in directed search. *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2004)* (pp. 415–422). Vienna, Austria.

Whittaker, S., Matthews, T., Cerruti, J., Badenes, H., & Tang, J. (2011). Am I wasting my time organizing email?: a study of email refinding. *PART 5 -------- Proceedings of the 2011 annual conference on Human factors in computing systems* (pp. 3449–3458). Vancouver, BC, Canada: ACM.