MetaDesk: A Semantic Web Desktop Manager

Robert MacGregor, Sameer Maggon, Baoshi Yan

Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, U.S.A.
{macgregor, maggon, baoshi}@isi.edu

Abstract

MetaDesk is an RDF authoring tool that facilitates entry of facts, rather than construction of ontologies. MetaDesk places no restrictions on vocabulary—users can invent terms on-the-fly, which the system converts into underlying RDF structures. Knowledge entry focuses on the creation of semantic structures that form scaffolding both for retrieving and interpreting facts. The most common hierarchic relationships turn out to be partonomies (whole/part structures) and set membership (as opposed to the traditional is-a hierarchies and class memberships). MetaDesk is also a semantic desktop that includes references to folders and documents within its knowledge base. We have found that the same semantic structures are appropriate for organizing desktop information

Introduction

A year ago we experimented with a tool for attaching RDF metadata to Web pages that used Protégé [Eriksson 1999] as the data entry (authoring) component. The tool required that a class be selected for instantiation as a prerequisite to knowledge entry. Our experiment was a failure, for two reasons. We found that the ontology-driven paradigm resulted in creation of artificial classes (often suffixed with the term "Annotation") that drew an artificial boundary between the objects being annotated and the metadata descriptions. Worse, it was just annoying—the effort to select a class before typing in an annotation discouraged use of the tool.[1]

In response, we invented a new tool, MetaDesk that makes RDF data authoring as quick and painless as possible. We use MetaDesk to record the kinds of metadata we generate during everyday tasks. We quickly discovered that the kinds of knowledge structures users (the authors, in this case) produced with the tool differ from the structures found in typical RDF databases. Currently, we are using MetaDesk as a personal information manager to keep track of projects, proposals, to-do lists, slides, etc., and as a launching pad for quickly bringing up specific folders and documents (like Windows shortcuts, only better organized and optionally possessing metadata annotations). Our intention is to add one or more additional knowledge sharing capabilities to MetaDesk, and then release it as a generic tool for managing information and for collaborating with other MetaDesk users.

Example: MetaDesk provides two metaphors for entering information—users can create "nodes" (represented internally as RDF resources) that are arranged in a hierarchy, and they can attach attribute-value pairs to nodes.

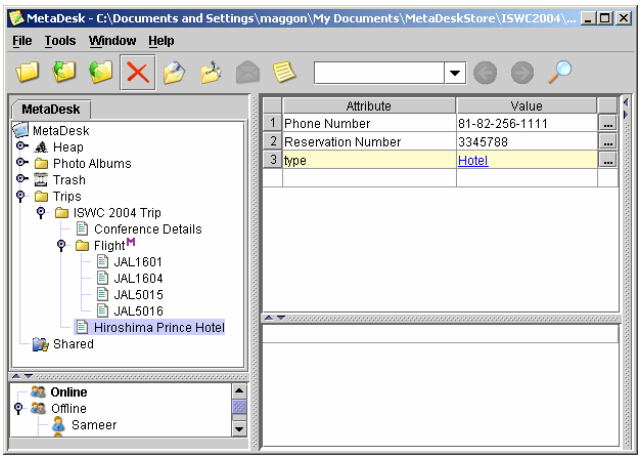

Figure1: Recording Trip information in MetaDesk

Suppose you are planning a trip to the forthcoming ISWC conference and you need to record information about the trip in an organized fashion. Details could include flight carrier, confirmation number, hotel preferences, prices etc. In addition, you would like the information to be represented in such a way that restructuring of the data is feasible. Storing such information in the current RDF authoring tools is a tedious process. As opposed to directly writing the information in the tool, you first have to create a myriad of classes and properties like Trip Class, Flight Class, and Hotel Class etc. Also, the domain and range constraints of the properties have to be specified. Further more, the ontological information is not very obvious in particular cases. For example, it is difficult to name the relationship between Trip class and Flight class and between Trip class and Hotel class. As a result, a naive user, or one in a hurry, would prefer to create such information in a text format than recording it in an ontology-driven RDF authoring tool. Our tool excels in simplicity, providing an efficient data entry paradigm.

[1] These artificial classes were created to provide domains for "annotation properties".

Recording the information in this example is easy and fast with MetaDesk. One can simply create a Trip node and add some child nodes to it. The child nodes could be a Flight node, a Hotel node and a Conference node. One can attach other information to individual nodes; for example, add a confirmation number to the Flight node. The resultant hierarchy is shown in Figure 1.

MetaDesk is all RDF-based--although users enter the data rather quickly without knowing anything about RDF, the created data is converted to RDF triples. Below we list the underlying RDF triples (in N3 format for readability) for the information shown in Figure 1. The "parentChild" links specify that under the "ISWC_2004_Trip" node are three nodes: "Flight" node, "Hiroshima_Prince_Hotel" node and "Places_to_Visit" node. Under "Flight" node are four other nodes representing individual connecting flights: "JAL1604", "JAL5016", "JAL5015", and "JAL1601". There are also RDF triples defining the reservation number and phone number for the hotel, etc.

```
myNS:Trips
  rdfs:label  “Trips” ;
  sew:parentChild    myNS:ISWC_2004_Trip .

myNS:ISWC_2004_Trip
  rdf:type  myNS:Trip ;
  rdfs:label “ISWC 2004 Trip” ;
  sew:parentChild myNS:Hiroshima_Prince_Hotel
                ,myNS:Places_to_Visit
                  ,myNS:Flight .

myNS:Hiroshima_Prince_Hotel
  rdf:type       myNS:Hotel ;
  rdfs:label "Hiroshima Prince Hotel" ;
  myNS:Phone_Number "81-82-256-1111" ;
  myNS:Reservation_Number "3345788" .

myNS:Places_to_Visit
  rdf:type sew:Desktop_Folder ;
  rdfs:label "Places to Visit" ;
  fileNS:fullpath "C:\\Documents and
Settings\\maggon\\My Documents\\Places to Visit".

myNS:Phone_Number rdfs:label "Phone Number".
myNS:Flight
  rdfs:label "Flight" ;
  sew:parentChild myNS:JAL5016 , myNS:JAL1604 ,
myNS:JAL5015 , myNS:JAL1601 .
```

Mapping MetaDesk to RDF

MetaDesk is represented as “triples all the way down”—every link in MetaDesk maps to a triple. A new node is created by highlighting an existing node, and explicitly typing the name of a child node, or by dragging something (a Web page, PDF file, Word Document, etc. or another node) onto the highlighted node. MetaDesk consciously imitates the gestures, look and feel used to construct hierarchies using Windows Explorer.

If ‘P’ is a node, and ‘C’ is one of its children, the link between them is represented by a triple of the form <P, R, C> where ‘R’ is either ‘parentChild’ or one of its subproperties. The ‘parentChild’ relationship is roughly definable as the most-general, directed structural relationship. As such it subsumes more specific relations such as whole/part, class/subclass, set/set member, or folder/subfolder. We originally assumed that it should also subsume the class/instance property (the inverse of ‘rdf:type’), but when viewing children of a class, we found that we wanted to see only its subclasses, not mixed in with its instances. A node can have multiple parents (it occupies the object position of multiple ‘parentChild’ triples). A special node called ‘Heap’ exists as a catch-all—an RDF resource that does not have a parent node is considered to be “on the heap”. This is handy for operations such as tabbed search that assumes that each node it displays is located somewhere in the hierarchy.

Each node N has zero or more attributes, represented by triples of the form <N, R, V> where ‘R’ is not a subproperty of ‘parentChild’ (or of its inverse). There are no restrictions on what attributes can be attached to a node (i.e., violations of domain constraints may be flagged, but are not forbidden). Users are encouraged, but not required, to fill in the attributed named “type”, which denotes the property ‘rdf:type’. A future version of MetaDesk will semi-automate the filling-in of type attributes.

RDF structures in their raw format are not readable, so we want to hide all details of RDF from users, including URIs and namespaces. Hence, all non-literal names that a user sees in MetaDesk (names attached to nodes in the hierarchy, attributes, and in attribute value position) correspond to RDF ‘labels’. Underneath, each label ‘N’ maps to a URI ‘U’, and MetaDesk asserts the triple <U, rdfs:label, N>. Some labels have semantics built in, e.g., “type” maps to ‘rdf:type’ and “parent class” maps to ‘rdfs:subClassOf’. By default, a label “xxx” that does not match an existing label is mapped to the URI ‘myns#xxx’, where “myns” is the URI for a user’s personal namespace.

An attribute value ‘V’ is stored as a literal (a string) if the relevant range information references a literal class (a subclass of ‘rdfs:Literal’), or as a resource if the range indicates a non-literal. If there is no range information, then the system first looks for a label matching ‘V’, creating a matching resource if there is. Otherwise, ‘V’ defaults to a string, but the user can convert a literal value it into a new resource (by gestures provided by MetaDesk) any time. Values representing brand-new resources are considered a part of the “heap”.

Importing Data

Arbitrary RDF files can be dropped into a MetaDesk hierarchy, but MetaDesk will not know which new resources to treat as nodes within the hierarchy. Instead, all of these nodes are assigned to the "heap". An exception is Class and Property resources. These are entered under the Ontology node, below either 'owl:Thing' or 'sew:Attribute'('sew' is the nickname for the namespace that is internally used by MetaDesk).

Arbitrary XML files can also be dropped into a MetaDesk hierarchy. These are automatically converted into RDF, with the top-most tag forming the root resource. The 'parentChild' Property is used to represent the relationship between tags and subtags (except when the subtag represents a literal). For example, for the following XML

```
<trip>
  <hotel confirmation="39880A78B">
  <flight fltnum="884"
          confirmation="S38BN04">
      <carrier>America West</carrier>
   </flight>
<trip>
```

Our translator would create resources of RDF type 'myns:Trip', 'myns:Hotel', and 'myns:Flight', with 'parentChild' links from the Trip resource to the Hotel and Flight resources. Each of the three attributes is converted into the obvious RDF triple. The Flight resource is linked via a triple to the string "America West" via a property named 'myns:carrier'.

Interaction with Windows Applications

The primary means provided currently for interacting with desktop objects are (i) drag and drop actions to/from the desktop and (ii) launching applications by double-clicking on nodes denoting them that reside in the hierarchy. Windows folders are a special case—when a Windows folder is dropped into the hierarchy, the corresponding MetaDesk node can materialize additional child nodes (on demand) corresponding to the contents of the folder when the node is "opened". Annotations attached to folders are persistent, but the 'parentChild' links that relate folders and subfolders are not stored persistently (to save space). Move and copy operations on folder nodes cause corresponding changes in the underlying Windows desktop hierarchy.

A complete semantic desktop should demonstrate similar levels of integration for other applications such as e-mail. Ideally, one or several commercial e-mailers could be integrated with MetaDesk. Alternately, one could mimic Haystack [Quan 2003] and implement an entire e-mail application (as a plug-in) within MetaDesk.

Plug-ins

MetaDesk architecture can be extended by using plug-ins to create alternate displays for the top and bottom panes to the right of the hierarchy pane. Plug-ins are associated with particular data types – when a node is highlighted, the default display plus all relevant plug-ins that correspond to the type of that node are presented as options. MetaDesk also enables users to select a *default* plug-in for the data type; this way MetaDesk remembers the user's choice for the next time. We have developed a photo viewer plug-in (Figure 2) that enables users to view the thumbnails of the images organized in MetaDesk. Whenever the user clicks on the Album Node (a node with the rdf:type – Photo_Album) in the hierarchy, the photographs are shown in the bottom pane. User can view as well as annotate the pictures thus embracing an interactive session.

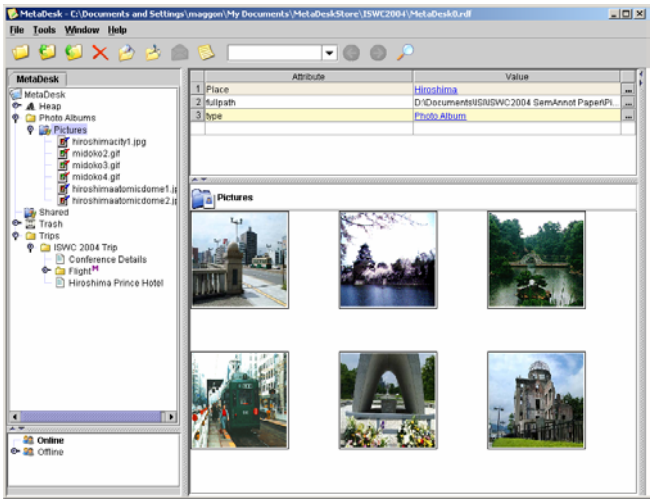

Figure 2: Photo-Plugin for displaying graphics resources

MetaDesk allows a user to choose the plug-in for any data-type (or class). For example, a user might want to associate the photo viewer plug-in with the nodes that have the type Photo Graphs instead of Photo Album. This leverages the ease of customizing MetaDesk according to personal preference. In addition to developing plug-ins for specific data types, one might consider writing a plug-in that enforces type restrictions on its input, or one that displays Protege-like templates in place of the free-form attribute editor that comes standard in MetaDesk. Such plug-ins would enable MetaDesk to mimic more traditional Semantic Web RDF editors. Thus, MetaDesk uses these plug-in points to keep track of the user's working behavior and provide self-personalization.

Ongoing Work

Search: Currently, MetaDesk supports keyword search. When searching for a match to the keyword "xxx", a triple <S, P, V> matches if one of S, P, or V has a label

containing "xxx" as a substring, or if V is a literal value that contains "xxx". Results may be in the form of a tabbed search, wherein each hit of the 'tab' key opens the hierarchy to the location of the next matching node, or the results may be placed under a newly-created search node which can further be annotated.

Ontology Alignment: Philosophically, MetaDesk runs completely against the grain by promoting "ontological promiscuity" and advocating bottom-up development of ontologies. "Promiscuity" refers to MetaDesk's encouraging users to make up their own vocabulary. In our scheme, we first let a thousand flowers bloom, and then specify semantic mappings (alignments) that say how one user's terminology relates to another's. We call this "grassroots alignment", since it empowers ordinary users to build terminologies, instead of requiring ontology experts. The current MetaDesk is missing two things: (i) "carrots" that encourage MetaDesk users to align their terminology with terms used by others and to fill in the type attribute on each node, and (ii) alignment tools that make aligning terms very simple. One example of such a carrot is a search facility that exploits alignments to increase the recall of its matches. Another is a report generator that produces denser, better organized reports when alignments are taken into account. ISI's WebScripter[Yan 2003] report generator incorporated both a carrot and an alignment capability into a single tool. Determining whether quality ontologies can be achieved bottom-up via a sufficiently mature set of carrots and alignment tools is at this point an open question—one that we believe deserves to be tested.

Future Directions

At present, we have hypothesized that end-user alignment can compensate for the ontological promiscuity engendered by multiple MetaDesk users, enabling a community of MetaDesk users to profitably share information. This hypothesis needs to be tested. Our near term goal is to add sharing capability, and then to distribute MetaDesk to a community of users. Our supply of "carrots"—tools that encourage end-users to align with each others' vocabulary—is still sparse. We will find out whether we are close to having a viable sharing infrastructure, or if more incentives are needed.

MetaDesk will eventually support multiple search regimens—more sophisticated ones will trade precision for user convenience (more typing yields more precision).

Conclusion

We have introduced MetaDesk, an original RDF authoring tool. MetaDesk's approach to RDF authoring is extreme: users immediately create metadata without defining ontology first. Instead, it is our belief that ontologies can be created later in a bottom-up fashion, as the by-product of creating and using data, rather than a straightjacket that inhibits the evolution of domain vocabularies. Compared with other ontology-driven RDF authoring tools (SHOE Annotator [Heflin 1999] OntoMat [Handschuh 2002] SMORE [Kalyanpur 2003] Melita [Ciravegna 2002]), MetaDesk is more ordinary-user friendly, more flexible in metadata creation, and provides immediate rewards to users' effort.

MetaDesk's metadata authoring paradigm allows quick data entry and organization. As a result, MetaDesk is already viable as a personal information manager. MetaDesk has been extended as a usable semantic desktop application. It is integrated with an actual user desktop, allowing direct annotations on file systems and direct launching of applications from within it. MetaDesk's simplicity in metadata creation as well as usefulness as a semantic desktop makes it a rewarding semantic web application.

References

F. Ciravegna, A. Dingli, D. Petrelli, and Y. Wilks. *Timely and Non-Intrusive Active Document Annotation via Adaptive Information Extraction*. Semantic Authoring, Annotation and Knowledge Markup, ECAI Workshop, July 2002.

H. Eriksson, R. W. Fergerson, Y. Shahar, and M. A. Musen. *Automatic Generation of Ontology Editors.* 12th Banff Knowledge Acquisition Workshop, 1999.

S. Handschuh and S. Staab. *Authoring and Annotation of Web Pages in CREAM.* WWW, May 2002.

Heflin, J., Hendler, J., and Luke, S. SHOE: *A Knowledge Representation Language for Internet Applications.* Technical Report CS-TR-4078 (UMIACS TR-99-71), Dept. of Computer Science, University of Maryland at College Park. 1999.

A. Kalyanpur, B. Parsia, J. Hendler, and J. Golbeck. *SMORE – Semantic Markup, Ontology, and RDF Editor.*

D. Quan, D. Huynh, and D. R. Karger. *Haystack: A Platform for Authoring End User Semantic Web Applications.* International Semantic Web Conference, Oct 2003.

B. Yan, M. Frank, Pedro A. Szekely, R. Neches, J. Lopez: *WebScripter: Grass-roots Ontology Alignment via End-User Report Creation.* International Semantic Web Conference, Oct 2003.