

Towards a Light-Weight Semantic Desktop

Knud Möller and Siegfried Handschuh

Digital Enterprise Research Institute, National University of Ireland, Galway
knud.moeller@deri.org, siegfried.handschuh@deri.org

Abstract. This paper shows that it is possible to build a system with Semantic Desktop functionality — the annotation and linking of desktop data across application boundaries, and the usage of such links to achieve new ways of organizing and finding data in a meaningful way — in a very simple and light-weight fashion. Instead of building an extensive and feature-rich system from scratch, we harness the power of modern, metadata-enabled file systems and search indices to create a prototypical system with minimal effort.

1 Introduction

The basic idea of this short paper is to present a light-weight architecture, that encompasses some but not all of the features associated with the term *Semantic Desktop*. Even though there exists no formal definition of a Semantic Desktop, a central idea is the linking of data in a number of ways:

- (i) **Linking data within the desktop** Objects on the desktop can be linked in arbitrary ways, so that data from previously isolated applications can be integrated and connected into a unified, meaningful graph. In this way, it would e.g. be possible to link entries from an address book to publications in a bibliographic database, by virtue of the fact that the people represented in the former are the authors of the publications in the latter. This would be possible even though the applications are unaware of each other's data.
- (ii) **Linking to data outside of the desktop** Objects on the desktop might also be linked to things that reside outside a particular user's desktop. E.g., the people in an address book might be linked to the countries they live in. Those countries are probably not available as data in any application on a given desktop, but there are a number of vocabularies, taxonomies and ontologies on the Web that represent the countries of the world. On a Semantic Desktop, arbitrary desktop objects can be linked to the resources in those and other ontologies.
- (iii) **Linking data between desktops** Some definitions also contain a social component, in that the Semantic Desktops from different users could form a network of desktops, in which they can collaborate and share and exchange data. This gives rise to the idea of a *Social Semantic Desktop*, as it is e.g. proposed in the European Nepomuk Project¹, which sets out to define a standard and reference implementation for Semantic Desktops.

¹ <http://nepomuk.semanticdesktop.org/>

Based on these linking capabilities, the promise is that users will have novel and more intuitive ways of organizing, finding and using their data, that are not restricted to hierarchical file structures or the boundaries of proprietary applications. Rather than starting from scratch, our claim is that an architecture that features some of those characteristics can be built by relying on and making use of technology that is already in place in current commercial operating systems. More precisely, we use existing metadata-enabled file systems and indices. Depending on the underlying operating system, an implementation that will go a long way can be built in a very simple and light-weight fashion, as the remainder of this paper will show. Furthermore, by tying directly into existing operating system technology and behavior, it is easier to design a powerful and easy-to-use user interface in which the user can quickly feel at home, and thus have a rich and unobtrusive user experience, which is a crucial factor for the widespread uptake of an idea like the Semantic Desktop.

2 Underlying Technology

The prototypical implementation that illustrates the ideas of this paper is based on Apple's Mac OS X operating system (OS X). Since version 10.4 (*Tiger*), OS X includes two technologies which are central to our approach: *extended file attributes* (or short "xattr") and the *Spotlight*² metadata index. Both will briefly be discussed in the following sections. Also presented will be *SpotMeta*³, a third party extension to Spotlight, which acts as a workaround to one of the shortcomings of Spotlight itself. For a more detailed discussion of xattr on OS X and Spotlight refer to a review by Syracuse [15].

2.1 Extended File Attributes

Extended File Attributes is a file system feature which allows to assign arbitrary metadata to file objects. This metadata is extended in the sense that it is not interpreted by the file system, unlike ordinary attributes such as access control or creation dates. Extended metadata could contain information like a document's author, the encoding format of a file, or other relevant information. xattr functionality is contained in many modern operating and file systems (though the names and implementations vary), such as ext2, ext3, ReiserFS and XFS (Linux), HPFS (OS/2), FAT and NTFS (WindowsNT), UFS (FreeBSD) or recently HFS+ (Mac OS X).

On HFS+, the extended attributes are stored directly as part of the file and have the form of attribute-value pairs. Attributes are strings and by convention follow a reverse-DNS naming style known from Java package declarations, values can be arbitrary data objects. Inherited from OS X's BSD heritage, manipulation is possible through a simple API, which contains `getxattr()`, `setxattr()`, `listxattr()` and `removexattr()` methods.

² <http://www.apple.com/macosx/features/spotlight/>

³ <http://www.fluffy.co.uk/spotmeta/>

2.2 Spotlight

An integral part of OS X, Spotlight is a live file system index which accepts complex queries on attribute value-based file metadata and returns a set of files that match the query. The building of the index is tied directly into the I/O part of the operating system's kernel, thereby guaranteeing that file system and index are never out of sync. Other parts of Spotlight are API support and interfaces such as the Spotlight search window (see Fig. 1). Since Spotlight only indexes the file system, applications that don't naturally handle their data in individual files (such as the Address Book or iCal) must generate dummy files for each entry they want to be indexed.

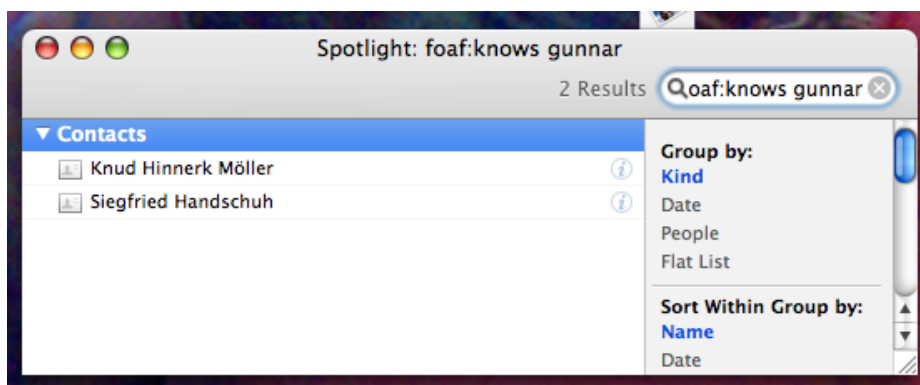


Fig. 1: The Spotlight search window, showing people who know Gunnar

Somewhat unexpectedly, Spotlight is not built on top of the xattr functionality of the file system. Instead, it uses so-called "Spotlight importers" to generate and harvest the metadata. Each importer acts as a plugin into the Spotlight system and is responsible for a specific type of file, such as PDF documents, MS Word files or PNG images. Also, each file will only ever be handled by one importer. These restrictions and choice of implementation have most likely been put into place to improve performance and ensure that system performance will not be slowed down: every time a file is touched, the responsible importer module will be activated to extract or generate the relevant metadata from the file and store it in the Spotlight index. Spotlight is extensible, and every new application can define importers for its data and add them to the system.

2.3 SpotMeta

It is tempting to implement the linking of data between different applications that is central to the Semantic Desktop on the basis of Spotlight. However, the

fact that Spotlight only allows to run one importer per file and ignores the extended file attributes makes this impossible at first — each importer would have to be modified or replaced, which is not a feasible solution. Spotmeta is an open-source project (not by the authors of this paper) that presents a workaround to this problem, by creating a bridge which will extract specific extended attributes from a given file and add those attributes and their values to the output of the Spotlight importer that is responsible for this file. In this way, it is now possible to not only add arbitrary metadata to any file, but also to include it in the Spotlight index. With all three technologies in place, we can now build a simple Semantic Desktop architecture.

3 Architecture

Looking at existing prototypes (see Sect. 4), any Semantic Desktop architecture needs a number of core components to facilitate basic functionality. If we ignore the social aspect, those components are (i) a *metadata store*, (ii) *application adapters* that access the various desktop applications and feed the metadata store, (iii) components that manage *linking* between objects and (iv) and *user interfaces* to glue everything together and allow the user to easily add and retrieve information.

3.1 Metadata Storage and Application Adapters

In our architecture, as depicted in Fig. 2, the role of the metadata store is played by the Spotlight Index, while the application adapter functionality is provided by the Spotlight importer modules. Note that those importers are the only way to have write access to the index — no direct write access is possible. Using the Spotlight index has some strong advantages, but also disadvantages. Because it ties into the I/O routines of the kernel, it is always up to date and in sync. Other approaches, which e.g. use an RDF (Resource Description Format⁴) store for metadata storage, will have to run an expensive background process to crawl the file system and gather metadata. Also, a Spotlight-based metadata store will immediately have access to a rich set of metadata, as it is common practice of every application provider to also provide matching Spotlight importers for their data. The index is also highly optimized and fast. On the other hand, it is only optimized for a specific kind of query. If we interpret the stored metadata as a set of triples (SUBJECT, PREDICATE, OBJECT) (this of course maps directly to any RDF-based solution and to the Spotlight file index by assuming SUBJECT = file, PREDICATE = attribute and OBJECT = value), then we can say that Spotlight only has a subject-based index. As a result, Spotlight only allows very restricted queries of the following form:

```
SELECT
  ?subject
```

⁴ <http://www.w3.org/RDF/>

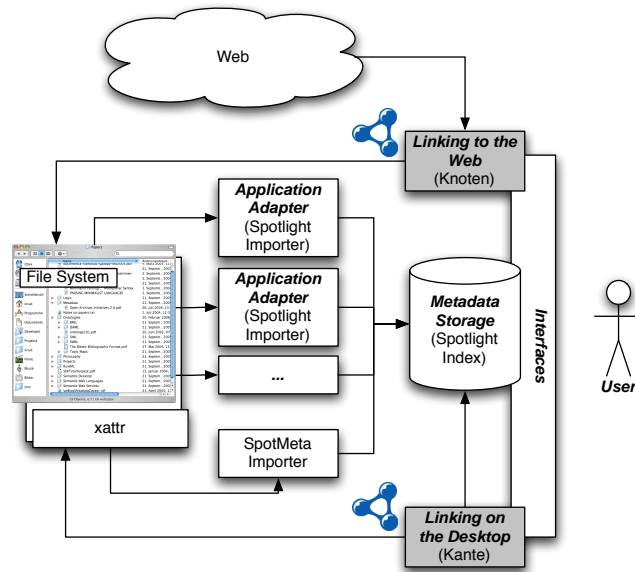


Fig. 2: A simple Semantic Desktop architecture, based on SpotlightTM

WHERE

(?subject [some predicate-object expression])

Therefore object-based queries such as “*Show me all people that Knud knows*”, which are core-functionality in RDF-based solutions, are not supported out of the box in our approach.

3.2 Linking on and Beyond the Desktop

So far we have shown how existing technology can be harnessed to achieve core functionality of a Semantic Desktop. Building on these components, we will now present two new applications that will provide the linking capabilities that were mentioned above, as well as a user interface to this functionality. Both are represented by grey boxes in Fig. 2.

Kante (German for “edge”) exposes functionality to easily link any two file resources with an arbitrary predicate⁵. The resulting triple is then added to the extended file attributes of the subject file, following a naming scheme that allows it to be picked up by the SpotMeta importer discussed in Sect. 2.3, and thus be added to the Spotlight index. In this way, the user can now make statements such as “*Knud knows Eyal*” or “*This picture was made by Benjamin*”. Figure 3

⁵ In the current version of Kante, predicates are simple strings. No namespace resolving occurs, even though some of the examples in this paper use a namespace notation.

shows how a user can annotate resources through the user interface of Kante. In the most basic way, they type the path to the desired subject and object files into the corresponding text fields. A much easier option is to *drag and drop* the resources directly from anywhere on the desktop and onto the image wells at the top of the Kante window. The drag and drop metaphor is used consistently throughout the OS X desktop, and this not only within the file system explorer (the “Finder”), but also in most other applications. Users are therefore able to drag contacts out of the address book, mails out of the mail client, events out of the calendar, etc. A final and very powerful annotation technique is to use the *autocompletion* functionality provided in the text fields of Kante. In the example in Fig. 3, the user had just entered “Nowack” into the object field and is now presented with a choice of the address book entry for a Benjamin Nowack or an email by Benjamin Nowack. These completion choices themselves are generated by performing a Spotlight query based on the typed word.

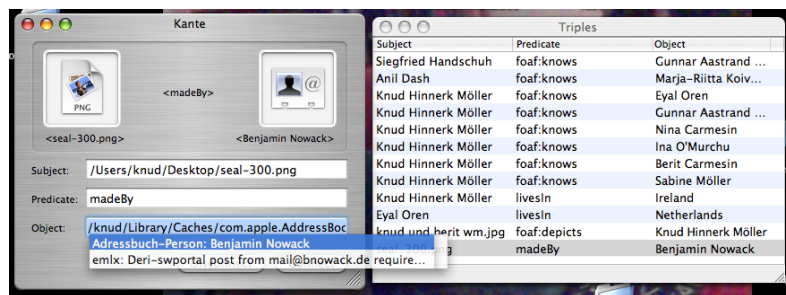


Fig. 3: Kante, a user-friendly UI for linking desktop resources

Once a new statement has been created and indexed in this way, it shows up in the “Triples” window on the right of the screenshot. The table in this window is bound to a Spotlight query which is constantly running in the background. From here, the user can access the triples’ subjects and objects by double-clicking them, which will open them in their respective application. However, the new information is not only accessible from within Kante, but from any application on the desktop that has access to Spotlight. E.g., the Spotlight search window shown in Fig. 1 illustrates how a user searches for and retrieves information that was previously generated in Kante, by typing “foaf:knows gunnar” into the search field. The results list shows that Knud and Siegfried know Gunnar.

Knoten (German for “node”) is another application, which accesses RDF-based ontologies or vocabularies from the Web and integrates them with the user’s desktop. More precisely, it extracts all resources from the ontology and makes them accessible to Spotlight. This is achieved by generating a file representation for each extracted resource and defining a Spotlight importer that knows how to extract relevant metadata from the file, so that it can be included

in the index. Currently, only those resources which have a defined `rdf:type` and a label (which can be any from a configurable list containing e.g. `rdfs:label`, `foaf:name`, etc.). Also, only the URI, type, label and potentially a depiction will be extracted and included in the metadata store; all other statements will be ignored. This is obviously an extreme reduction of potentially very rich data contained in an ontology. However, it suffices to allow the user to then make statements about those entities which were extracted.

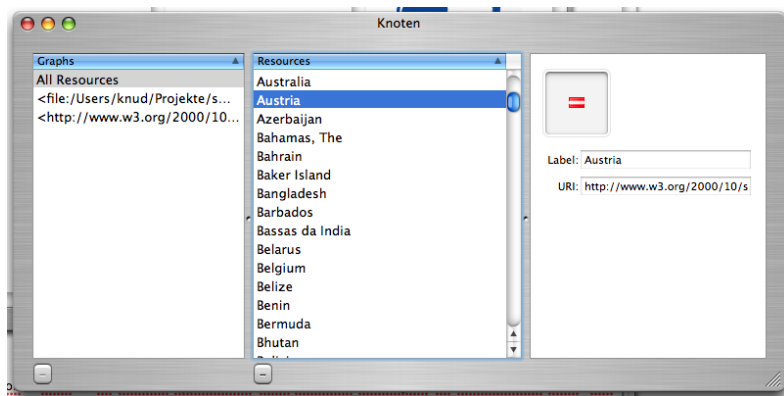


Fig. 4: Knoten, an application for integrating ontologies with the desktop

Fig. 4 shows the user interface of Knoten. The user has downloaded an ontology of countries⁶, and Knoten’s Spotlight importer has made the resources available to the metadata index. Having integrated the country ontology, the user can now make statements such as “*Knud lives in Ireland*”. The example also illustrates how Knoten can integrate data from different ontologies, namely by merging information about resources with the same URI. In this case, the user has downloaded a second ontology that makes additional statements about each country by assigning them a depiction.

3.3 Escaping the Desktop

Our approach does not consider social or collaborative aspects of Semantic Desktop architecture. However, a simple suggestion for outward communication that makes use of the new functionality can be made. The semiBlog tool for Semantic Blogging [8,7] implements functionality to generate blog posts that are annotated with data from various desktop objects, such as address book contacts or calendar events. Based on a plugin system, semiBlog can extract data from other applications and translate it to RDF, which will then be attached to the blog

⁶ <http://www.w3.org/2000/10/swap/test/dbork/data/Country.daml>

post. Once it has been published to the Web in this way, the data has “escaped” the desktop, and is available for sharing and reuse by others.

To access the data and enable UI features such as autocompletion, semiBlog uses the Spotlight API in the same way as Kante, which would make it easy to integrate into our approach. Currently only individual desktop objects are handled, but it will be easy to let semiBlog also take into account the inter-application links that have been established using our approach to a Semantic Desktop architecture.

4 Related Work

The term “Semantic Desktop” was originally coined by Decker and Frank [4]. However, the earliest vision of a such a system — even though it was not called that at the time — was of course Vannevar Bush’s often cited idea of the Memex [2], which was never actually built, but had a strong influence on the later development of concepts such as hypertext by Nelson [9] and Engelbart [5]. Gifford et al.’s paper on semantic file systems [6] describes in great detail a possible implementation of some of the Semantic Desktop’s core ideas, such as extraction of metadata from files and associative access (as opposed to access via a tree-based hierarchy). In great parts it reads as though it described Spotlight or similar metadata-enabled file indices, e.g. by saying that a semantic file system “*is an information storage system that provides flexible associative access to the system’s contents by automatically extracting attributes from files with file type specific transducers*”.

Search systems similar to Spotlight are Beagle⁷ and Google Desktop⁸. Both Spotlight and Beagle retain live indices, that are notified from kernel level when the file system changes. Google Desktop does not retain a live index.

Recently a number of systems that implement a more extensive suite of Semantic Desktop functionality, rather than only search, have been developed. The Gnowsis [14] system is a Java-based platform that allows linking of desktop data according to user-defined *Pimos* (Personal Information Model Structures), and stores the generated metadata in a central RDF store. Additional metadata is extracted from desktop objects through Aperture⁹ adapters, which play a similar role as the “*file type specific transducers*” mentioned by Gifford or the Spotlight importers used on OS X. Haystack [12] is both a SW data browser and “*universal information client*”, that is extensible through plugins, but does not interact naturally with other applications. Similarly, IRIS [3] is an application framework to allow the development of Semantic Desktop applications, and comes with a number of knowledge management applications, such as an email client, office applications or a calendar. DeepaMeetha [13] is a knowledge management platform that organizes, visualizes and gives access to desktop objects based on the Topic Map paradigm [1]. All four Semantic Desktop projects clearly go beyond

⁷ http://beagle-project.org/Main_Page

⁸ <http://desktop.google.com/>

⁹ <http://aperture.sourceforge.net/>

what is presented in this paper. Also, all have in common that they are built largely on top of and independently of the underlying file and operating system. To a large extent, they try to replace existing functionality, instead of integrating it. This gives them great flexibility and power, but also makes it harder to adopt for users, who are used to a particular desktop environment and particular applications. On the other hand, the approach presented in this paper tries to achieve less, but by only requiring a minimum of change and effort from the user, we think it more likely that Semantic Desktop-like technology will be accepted and used.

As a last reference to related work, Oren gives an extensive overview of Semantic Desktop and relevant information management and knowledge work literature [10].

5 Conclusions and Future Work

In this paper, we have presented a simple and light-weight architecture, which implements two core features commonly associated with the term *Semantic Desktop*: (i) linking data within the desktop, across application boundaries and (ii) linking to data outside the desktop. At the core of this architecture is existing desktop technology: the extended file attribute technology common to many modern file systems and the Mac OS X Spotlight file search technology. While other features of the Semantic Desktop, such as e.g. the social and collaborative aspect, have not been considered, we have shown that it is possible to achieve a lot of the desired functionality with only little effort.

As future work, we will explore possibilities of introducing the stack of Semantic Web technologies into our architecture. This could e.g. be achieved by defining a interface from Spotlight to the RDF query language SPARQL¹⁰, or to ActiveRDF [11], a library for mapping RDF-based data to an object oriented model in the Ruby programming language. Also, the functionality contained in Kante and Knoten is currently restricted to these applications — it would be beneficial to expose it in the form of a library which can then be used by other applications. Only in this way could a real Semantic Desktop be built.

A possible drawback of relying on the technology of a particular platform, rather than re-implementing functionality in a platform-independent way, is of course that it cannot be applied universally straight away. Furthermore, because we are relying on an existing and (mostly) closed architecture, we have to accept the technology as it is. We cannot, for example, decide to improve the query capabilities of the system by adding predicate and object indices to Spotlight. A system that is built from scratch on the other hand would have these options. However, we think that the simplicity of our approach and the fact that tighter OS integration offers a better overall user-experience makes up for these drawbacks. Also, the approach is sufficiently simple to be portable to other platforms that offer extended file attributes and a live metadata index such as Spotlight.

¹⁰ <http://www.w3.org/TR/rdf-sparql-query/>

Acknowledgements

This material is based upon works supported by the European Commission under the Nepomuk project FP6-027705.

References

1. M. Biezunski, M. Bryan, and S. Newcomb. ISO/IEC 13250:2000 Topic Maps: Information technology — document description and markup languages. Technical report, ISO/IEC, December 1999.
2. V. Bush. As we may Think. *The Atlantic Monthly*, July 1945.
3. A. Cheyer, J. Park, and R. Giuli. Iris: Integrate. relate. infer. share. In *1st Workshop on the Semantic Desktop at ISWC2005*, Galway, Ireland, November 2005.
4. S. Decker and M. R. Frank. The Networked Semantic Desktop. In *WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web*, 2004.
5. D. C. Engelbart. Augmenting human intellect: A conceptual framework. Summary report, Stanford Research Institute (SRI), Menlo Park, CA, USA, October 1962.
6. D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. James W. O’Toole. Semantic File Systems. In *13th ACM Symposium on Operating Systems Principles*, October 1991.
7. K. Möller, U. Bojars, and J. G. Breslin. Using Semantics to Enhance the Blogging Experience. In *The third European Semantic Web Conference*, pages 679–696, Budva, Montenegro, June 2006.
8. K. Möller, J. G. Breslin, and S. Decker. semiBlog - Semantic Publishing of Desktop Data. In *14th Conference on Information Systems Development (ISD2005)*, pages 855–866, Karlstad, Sweden, August 2005.
9. T. H. Nelson. A file structure for the complex, the changing, and the indeterminate. In *ACM 20th National Conference Proceedings*, pages 84–100, Cleveland, Ohio, 1965.
10. E. Oren. An overview of information management and knowledge work studies: Lessons for the semantic desktop. In *The 2nd Workshop on the Semantic Desktop, at ISWC2006*, Athens, GA, USA, Nov. 2006.
11. E. Oren, R. Delbru, S. Gerke, A. Haller, and S. Decker. ActiveRDF: Object-oriented semantic web programming. In *16th International World Wide Web Conference (WWW 2007)*, Banff, Canada, May 2007.
12. D. Quan, D. Huynh, and D. R. Karger. Haystack: a Platform for Authoring End User Semantic Web Applications. In *Second International Semantic Web Conference (ISWC2003), Proceedings*, 2003.
13. J. Richter, M. Völkel, and H. Haller. Deepamehta - a semantic desktop (poster). In *1st Workshop on the Semantic Desktop at ISWC2005*, Galway, Ireland, November 2005.
14. L. Sauer mann. The Gnowsis — Using Semantic Web Technologies to build a Semantic Desktop. Master’s thesis, Technische Universität Wien, December 2003.
15. J. Syracuse. Mac OS X 10.4 Tiger, April 2005. <http://arstechnica.com/reviews/os/macosx-10.4.ars/>