Knowledge Organization Mashups

Erik Wilde
ETH Zürich (Swiss Federal Institute of Technology)

March 2006 (TIK Report 245)

Available at http://dret.net/netdret/publications#wil06f

Abstract

Information today is often distributed among many different system within a complex IT environment. Using this information for creating knowledge organization systems and services thus involves using this distributed information and re-purposing it within new applications. The current trend in Web technologies to build systems not in a monolithic fashion, but rather intended as building blocks within a constantly evolving and unplanned landscape of information processing agents. This approach can be used as a foundation for building *Knowledge Organization Mashups*. We investigate the possibilities and challenges of this type of application, and as a case study describe a service for managing bibliographic metadata.

Contents

1 Introduction

Knowledge is as much about facts and their categorization as it is about making connections between facts which lead to interesting and useful new perspectives on facts. Following the trend towards opening up Web-based systems not only to users but also to other systems, we discuss what we call *Knowledge Organization Mashups*, a term derived from the Web trend of *mashups*, which are Web applications using and recombining information from other applications.

In a way, knowledge organization systems and mashups could be regarded as something rather different, because knowledge organization systems often use a complex and highly specialized data model, while mashups often are based on putting together rather simple building blocks of information extracted from various sources.

On the other hand, there is a close connection between knowledge organization systems and and the idea of mashups, because the goal of both approaches is to provide a useful service by combining information that has previously been unconnected. In the first case, the approach is build a complex system for doing this, while in the second case, the idea is to create an environment where components can evolve based on new ideas for recombining them.

In this paper we concentrate on the synergy between the two fields, elaborating on how knowledge organization mashups could be built, why they are a useful approach to tackle the ever-growing complexity of today's IT systems, and presenting a case study of a system which has been built based on the ideas presented here.

2 Mashups

The Web is the biggest information system ever built, and some claim it is also the greatest information system ever built. At least, it is a proof of the fact that very complex information systems can be built, and it may also be a proof of the fact that systems of such magnitude can only exist if there is room for evolution. One of the critical success factors of the Web is its architecture [13], which is built on top of very few and basic principles, and apart from that leaves a lot of room for new developments and unforeseen combinations of existing technologies.

One of the fundamental properties of the Web is that once information is made available, it can be reused not only by a browser for rendering a Web page, but also by machines for any other purpose. While this principle has caused many legal (for example, lawsuits about "deep linking") and technical (for example, Web sites being overloaded by crawlers) problems, the benefits of this property far outweigh the problems. Legal problems have been tackled by updated laws or by simply better configuring the Web servers, so that misuse is prohibited through technical measures. Most Web sites today are grateful to be crawled and thus to become part of a search engine index, and if they really prefer not to be crawled, the robot exclusion protocol can be used.

Many recent trends in Web technology have been summarized under the term *Web 2.0*, which does not define or identify anything specific. One of these trends is an increasing amount of *mashups* (sometime also referred to as *Web application hybrid*) being built. The reason for this that an increasing amount of information is available on the Web, and more importantly, the amount of *content* (as opposed to HTML-encoded *presentation* information) has been growing rapidly. This makes it much easier to reuse information, because using information encoded in *XML* [5] is much easier than retrieving HTML and then trying to extract the content from it.

While the availability of content has been one of the driving factors for the popularity of mashups, so is the availability of technologies and tools to develop applications based on this content. Most content is available in XML, but even though XML is easy to understand, tools are required to efficiently process this content. In general, tools for processing XML have become standard building blocks of many application development environments, the most basic building blocks being APIs for accessing XML such as *DOM* [16] and XSLT [14] as a dedicated programming language for transforming XML.

On the Web-specific side, trends such as *Asynchronous JavaScript and XML (AJAX)* allow the dynamic access to any XML-based data source out of standard Web applications. They even allow the dynamic refreshing of content in response to user interactions, which breaks the traditional limitations of static Web pages only being reloaded on refresh or link traversals.

To summarize, mashups are a way to reuse the increasing amount of information being available on the Web; as well as a trend for supporting tools and technologies to make application development in this environment easy. Apart from these technological issues, it is also important to see mashups as a new class of applications, breaking the traditional approach of standalone or closely-coupled distributed systems. Mashups are loosely coupled distributed systems. They are loosely coupled to the extreme, in a way where the developers of the individual components do not know each other and possibly do not even know that their application is being used as a component by another application. Naturally, this includes the risk of breaking applications, but in an environment where mashups provide an added value, out of mutual interest some kind of coordination will happen which prevents these systems from collapsing.

Mashups can be compared to a distributed system architecture often called an *Information Bus* [18]. Mashups are less planned and coordinated than Information Bus systems, and thus are more dynamic in their development and evolution, and less robust against changes of components. Interestingly, the four core design principles of the Information Bus architecture (core communication protocols have minimal semantics, objects are self-describing, types can be dynamically defined, and communication is anonymous) also apply to the current Web architecture and mashups as a typical example of applications being used in this environment.

3 Knowledge Services

In any complex environment, it is unlikely that it is possible to design and build one unified knowledge organization system or service which will satisfy all requirements and thus be used by all people. Instead, systems and services often evolve out of practical needs or are simply realized in a certain way for historical reasons. Instead of approaching such a fragmented and heterogeneous landscape with the goal to unify it and create one self-contained system containing all the information, the more practical approach is to let the systems evolve, and then provide the means to use them easily to create new and useful services.

3.1 Knowledge Service Evolution

One of the most interesting and challenging facets about knowledge is that it is constantly changing, and that any formalization of knowledge and facts will be obsoleted at some point in time. This always was and still is a huge challenge for the knowledge formalization community, and in almost all productive systems using knowledge organization techniques, a static ontology is assumed. Many ontology models allow growing ontologies, but in most cases this in only allowed in a monotonic fashion, so that no older data can be invalidated by ontology extensions.

The bigger a knowledge organization systems and the underlying formalization is, the less likely it is that it will be completely re-designed to better adapt to the knowledge which may have changed since the system was built. As with any complex and monolithic IT system, the tendency to keep this system and slightly adapt it to new requirements is much bigger than with more agile and flexible component-based systems, where individual components can be replaced and updated much easier.

So we propose that the mashup model presented in Section 2 also could be a beneficial approach for building knowledge systems and services, and the implications of this are discussed in detail in Section 4. One of the main challenges of such an approach is the question how to coordinate the evolution of knowledge services in the sense that there is some shared understanding of the different domains covered by the different components. Different approaches to solve this problem have been presented, for example one being called the *Knowledge Bus* [19], in an analogy to the *Information Bus* mentioned in Section 2. However, many of the approaches assume that there is a common and shared ontology, and this may not be the case in a scenario where evolution is left to its own devices.

Another possible way to go is the adoption of *Semantic Web* technologies, which are based on the assumption that different applications use the same model for representing semantics, but without the need to have one unified model encompassing all application models.

3.2 Semantic Web Technologies

The *Semantic Web* was a vision initially developed [2] to enable machine-readable data being available on the Web. The approach was very Web-like: Develop a technology to make this possible, and then applications will be built on top of this and further developments can be made. The basic idea of the Semantic Web is that resources on the Web are described using *RDF* [15], and that these descriptions are based on ontologies in languages such as *RDF Schema* [6] or the newer and more powerful language *OWL* [21].

One of the key features of Semantic Web technologies is their ability to provide a level of abstraction that is ideally suited for the information required in knowledge organization systems. While the exact format and structure of data being provided by different services or applications may differ, the semantic description of the data makes it possible to encode the fact that the underlying meaning is the same. Thus, applications being built on top of Semantic Web technologies can be more robust against changes in the data structures, or they can detect when the data structures have remained the same, but the semantics have changed.

This kind of ideal scenario assumes that the information provided by components in a heterogeneous IT infrastructure is marked up semantically. While this would be a very good starting point for building knowledge organization systems based on these components, in reality only very few services today provide semantic information. Most services provide data structures encoded in XML, and the element and attribute names may carry some semantics, but often this is only a fraction of what would be required to fully understand a component's interface.

The Semantic Web is a vision that will only slowly become a ubiquitous reality. But only if existing applications based on Semantic Web applications are convincing examples of what added value can be provided, and technologies and tools are available to build new applications, will the Semantic Web become a success story. Today, working with Semantic Web technologies is much harder than working with XML technologies, and therefore only few applications outside of the research world are built using Semantic Web technologies.

The idea of mashups presented in Section 2 is based on the assumption that components which can be recombined to build new services are easily usable and easy to understand. At the current state of technology this means that services should provide a rather simple, XML-based interface, which can be used by anybody familiar with XML technologies. Optionally, Semantic Web technologies could be used to complement the XML data with semantic information, but the service should be designed in a way which makes it usable without relying on the semantic information.

This principle of being thoughtful about “naïve” users even if advanced technologies are being used is essential for supporting the evolution of knowledge service mashups, because it lowers the barrier-to-entry. Section 4 investigates the different facets of this design principle in more detail.

4 Towards Knowledge Organization Mashups

Since we propose to view the evolution of a set of knowledge organization systems and services as a realistic goal, the question is how this evolution can be planned or at least supported. By definition, an evolutionary process can never be fully planned, and promising development paths will sometimes turn out to be dead ends, and vice versa. However, when thinking about knowledge organization in large organizations, it is possible to influence the environment in a way which will be more or less supportive to the evolution of knowledge organization mashups. The following section review the possibilities to support this evolution from a technical, design, and organizational perspective.

4.1 Technical Issues

IT systems in large organizations tend to develop in a rather uncontrolled way, depending on how critical the IT systems are and how big the organization's interest is to be able to interconnect all components. In commercial organizations depending heavily on the IT infrastructure, strict guidelines are enforced for the development of every new component. For example, in many banks the complete IT infrastructure has been built around CORBA as the middleware for distributed systems. Such a decision ensures the development of a homogeneous service landscape, but also makes the development of new components very expensive and the migration of individual components to other technological foundations impossible. Newer approaches such as *Service Oriented Architectures (SOA)* will avoid some of the costs associated with complex middleware infrastructures, but still they require a lot of internal standardization to manage the set of services.

In other organizations where the IT systems are not only mission critical resources, but also used for additional purposes, the rules for the development of this landscape can be relaxed. In universities, for example, there are usually some core services such as human resource management and the management of teaching and research activities. However, there are also many supplementary systems and services which do not belong to the indispensable core of the IT landscape, but instead are being created and managed by individuals and groups within the university.

We propose to establish a set of guidelines for developing these systems and services in a way which makes them accessible for other users within the organization. These guidelines have to be concise and easy to follow, otherwise people will simply ignore them. Ideally, there should be some way of supporting or even rewarding people following these guidelines, for example with introductory courses, toolboxes, help desks, and directories where these services are being listed. The following points are the essence of what these guidelines should define:

- *Data Model:* A system or a service should have a well-defined and well-documented data model. The exact form of the data model and the exact form of documentation is up to the developers, but there must be some minimal way for users of the system or service to understand the semantics of the data and operation being implemented by the system.

- *Data Format:* In addition to the data model, there should be a well-defined mapping to a data format, which can be used to interact with the system. Ideally, this data format should be based on XML and described by some schema, the *XML Schema* [23, 3] language should be the preferred language for doing this (of course, some data should not be structured using XML but instead use dedicated binary structures, for example non-textual media content such as pictures, audio, or video).

  The data format should be "well-designed XML", which means that it should be designed in a way that it can be used by others using basic XML technologies. This is an important point because many systems internally use some data model which is then simply mapped to an XML format by using some export functionality of the data modeling software being used. Typical examples of this are XML Schemas generated from UML models. While these XML Schemas are technically correct and make it possible to exchange information via XML, it is often very hard to use this data for anything else than re-importing it into the UML-based environment through generated import code.

  The question of how to align data models and "well-designed XML" (which may require some human intervention rather than being generated automatically from some data model) is an unsolved question so far [27], so while the guidelines here ask for an ideal co-existence and co-evolution of a data model and the associated XML schema, this cannot be easily achieved with today's technologies and tools.

  Another format worth mentioning here is RDF, which also could serve as a data format for some data model. Even though RDF's XML syntax is a well-designed XML format, it is not easy to access using basic XML technologies, so it should not be considered a good candidate as a data format (see also Section 3.2 for more information about Semantic Web technologies).

- *Data Access:* Apart from making data models accessible through an easy-to-use data format, there also must be access to the data in simple way. The generally accepted protocol for accessing data today is *HTTP* [10], which has gained its popularity as the access protocol of the Web. In the simplest form, HTTP can be used to implement so-called HTTP Web Services, which are services which are accessed through simple HTTP `GET` or `POST` requests and then send back XML as the result of the request.

  For more complex and more formally modeled Web services, the *SOAP* [12] protocol can be used. In this case, the transport protocol still is HTTP, but the provided services are described using *WSDL* [7], which defines the interfaces in terms of XML Schemas for input and output data. The data is then being sent using SOAP XML, which contains service-specific XML. While SOAP provides a more powerful environment for describing and implementing Web services, it also adds a layer of complexity, and generating and processing SOAP XML without the support of a SOAP toolbox is a tedious task.

These guidelines do not really prescribe and specific technology (apart from XML and HTTP), but they put the focus on the simplicity of services. The simpler a service is designed, the easier it is for other to use this service and integrate it as a component into a new service.

For example, if a service produces well-designed XML, then it is a fairly easy task to use XSLT to process this XML and extract the required information. If the service produces XML described by an XML Schema, this processing can be even done in a type-oriented way, which facilitates the processing even further. If, on the other hand, the XML that is being produced has been generated from some underlying data model which is poorly documented, processing it can be cumbersome or almost impossible.

The general question of how XML has to be interpreted to be understood by users is a yet unsolved question, and thus these guidelines can give no definitive answer. The Semantic Web provides one possible solution to this problem, and one possibility to merge "plain" XML and Semantic Web descriptions would be to define a mapping of the XML to RDF [22]. This would on the one hand enable "naïve" users to use the XML, and Semantic Web users could transform it into RDF. However, how this transformation could be derived from a data model which is not necessarily using RDF as its foundation is an open question.

As pointed out in Section 3.2, ideally Semantic Web technologies could serve as the foundation of the technical guidelines presented here. Currently, however, the state of technology and tools make it more advisable to rely on XML as the foundation. For example, while XSLT and the upcoming XQuery [4] language are advanced and widely supported technologies for extracting data from XML, the RDF equivalent *SPARQL* [20] is not very widely implemented, and only very few application developers have any experience with it.

The technical guidelines presented here are just a simple outline of which principles and technologies should be used for exposing a service inside an organization. The amount of regulation that is appropriate for implementing these guidelines inside an organization depends on the freedom that the development of services should have. Less stringent rules will produce more services but result in less homogeneity, while more stringent rules will stifle the realization of some good ideas, but the remaining services will be more homogeneous and thus easier to use.

		Output	
		HTML	XML
Input	URL-encoded	HTML Forms (`GET`) OpenURL	HTTP Web Services RSS/Atom
	XML/MIME	HTML Forms (`POST`) XForms	SOAP Web Services

Figure 1: Different Input/Output Types of Components

4.2 Design Issues

The technical guidelines described in the previous section provide some guidance to how services should be implemented. In addition to these technical guidelines, it is also important how services are being designed. It is possible to expose the same functionality of a service in very different ways, and this has important consequences for the ways in which a service can be reused as a component.

One example for this is the *OpenURL* [1] standard, which specifies how to encode metadata in the query string of a URI. Many libraries provide OpenURL access to their catalog, in the sense that anybody may access the OpenURL server with a request for catalog data. However, in most cases the result is only provided as an HTML page, which means that the OpenURL service can be used as a convenient entry point into a library's catalog, but it cannot be used as a component of another service. In such a case, the OpenURL service can be referred to in the user interface of a new service (simply by generating OpenURL links), but its results cannot be integrated into a new service.

This is one of the basic question of how hypermedia systems should be designed: Whether existing services should be accessible through different interfaces, or whether there should be only one interface for each service. In the context of mashups, where services are being recombined, this question must be answered in the way that services should be reusable in the context of other services, allowing the filtering, processing and extension of the data provided by the embedded service.

The question of how services should be designed to provide the best reusability is an important one, and Figure 1 provides an overview of some of the possible technologies for deciding how input and output of a service should be implemented.

The question of how to implement the input side of a service (i.e., the question of how to send requests) should be decided based on the complexity. For simple services, a HTTP Web Service based on URL-encoding the input parameters is usually sufficient. For more complex parameters (typically, anything that is structured and cannot be easily represented in the string-based form required by URL-encoding), it usually makes more sense to provide the parameters as body part of the HTTP request. A good example for this are HTML forms: While simple forms can be implemented using URL-encoding the parameters in HTTP `GET` requests, more advanced forms containing complex parameters such as file contents must be implemented using HTTP `POST` requests, which encode the parameters in *MIME* [11] syntax (in this case, no XML is being used).

For the output side, the question of whether HTML or XML should be generated is a question which decides whether the service can be integrated as a Web page, or as an application. Only if the results are provided in XML, the integration into other services is possible. In many cases, it could be a good idea to implement both output formats. Theoretically, a service could always produce XML and associate a stylesheet with it [8] and then let the service user decide whether to apply this stylesheet or not, but since not all browsers handle this correctly, this is not a realistic solution. Instead, the required output format could be one of the service's parameter, and this parameter would then simply decide whether the XML → HTML transformation is applied or not.

The choice of whether a service's XML or HTML output is used then is at the discretion of the service's user. This decision is similar to the basic choice in hypermedia between *transclusion* and *hyperlinking*. While transclusion presents the original data in a new context, hyperlinking refers to the original data and the data is always presented in its original context. In case of mashups, the choices are as follows:

- *Reusing Information from another Service:* In this case, the new service reuses the XML received from another service. This can be done on the server side of the new service or even on the client side (if AJAX or a similar approach is used), which makes it possible to seamlessly integrate the information from multiple services.
- *Redirection to another Service:* If a service only provides HTML output, the seamless integration is harder. In the simplest case, a link is created which then takes users to the results generated by the other service. More subtle techniques are HTML `FRAME` or `IFRAME` approaches, but these are often rather unflexible and hard to be designed as being reusable in a generic way.

Using these guidelines, it is possible to design a service with the maximum amount of openness, extensibility, and flexibility. For example, by always designing a service to produce XML, and adding HTML only as a optional last step of the output pipeline, it is much easier to concentrate on producing the result, and it is also much easier to concentrate on simply changing the formatting of the result, or to add an alternative formatting of the result.

When designing services, sometimes it is simple a question of combining existing technologies in unusual ways. For example, content syndication technologies such as *RSS* [28] or *Atom* [17] most often are used to distribute HTML content. However, they could also be used to distribute XML content, thus moving from the presentation-oriented syndication applications of today to content-oriented syndication.

4.3 Organizational Issues

The technical and design issues discussed in Sections 4.1 and 4.2 provide a way to encourage the implementation of services which can be reused, but the question remains how to define, distribute, and enforce these guidelines inside an organization where people have the liberty (and the right) to ignore them.

The most important aspect is that building components which can be reused in mashups is not about instant gratification, but about participating in a community where every participant contributes, and all participants benefit. Thus, community-building is essential, and this can be done through bringing together the participants in a way which supports their activities.

Unfortunately, there is no formal framework for how this could be done, nor is there a technology which is ideally suited to accomplish this. There are, however, approaches which solve at least part of this problem. For example, for Web Services the *UDDI* [9] registry can be used as a directory for collecting information about available services. However, UDDI is geared towards WSDL/SOAP-based Web Services, which can be limiting in an environment where the majority of services is using different technologies. Another possibility is to use Namespace descriptions [25] of the schemas used by services, and to use these namespace descriptions as a starting point for compiling a directory of namespaces, schemas, and services.

Regardless of which technology and strategy is being used, it is important that the organization actively promotes and assists the development of individual services and the services landscape. If this is not done, it is very likely that fragmentation occurs within the organization, and that individuals or groups within the organization develop services without regarding themselves as part of an evolving services landscape.

On the global level, this trend towards communities of users being interested in mashups can already be observed, for example with Google's *Google Maps* service, which is designed to be used as a component in Web-based applications. By providing an easy-to-use interface which can be used by Web developers in a minimum of time, a new way of integrating geographical information with localized services (which sometimes is extracted from external services as well and then combined in a mashup with the maps) has been made possible.

5 Case Study

As a case study of the issues discussed so far, the *Shared References (ShaRef)* [24] project is discussed. It is a project with the goal of building a service for managing bibliographic data in a collaborative research environment. The initial idea was to replace the individual EndNote and BibTeX files of many researchers which a collaborative service which would make it easy to share this information with others. As an added benefit of these sharing capabilities, they should be extended with publishing features to make this data available to other applications, too, for example the Web content management system which should be able to include publication data directly with a research group's Web page.

5.1 Data Model and Format

One of the most important goals of the project was to create a tool which could be easily understood and used by researchers to support them in their routine tasks. Thus, we decided to define a data model which is simple and reuses many of the concepts researchers already know from using EndNote or BibTeX. The data model is geared towards end-users, which means that from the librarians point of view, it lacks some features which would be considered essential.

Apart from the data model for individual bibliographic references, it also is possible to create associations between references, and to assign keywords. The semantics of these associations and keywords, however, are not defined by the data model, they can be typed by users and any semantics then can be assigned to them. This way, the data model can represent network-like structures layered on top of a set of references, and when importing references from a source providing this kind of association information, it can be mapped to the respective facets of the data model.

The data model and data format have been based on XML, because the whole system has been built in an XML-centric manner. This way, there was no non-XML data model which had to be mapped to an XML Schema. Instead, the data model is defined by the XML Schema and additional annotations about semantics and constraints which cannot be expressed in XML Schema.

5.2 Interfaces

The interfaces have been designed in a way which should make it easy to integrate the system into an existing landscape of systems and services. The XML format is the foundation for import and export features, which thus enable the lossless round-tripping of data. Additional bibliography formats are supported as import and export formats, which make it easy to migrate from and to other applications.

With the exception of BibTeX import, which requires a specialized parser because of the idiosyncratic syntax, all import and export filters are implemented in XSLT 2.0, and all options which can be set for import or export are described declaratively in XML files. Because of this design, new import and export formats can be easily integrated into the system, as the user interfaces are generated from option descriptions and the filters are executed using the XSLT processor.

Apart from filters for import and export from and to other systems, there also is an HTML export filter which enables users to directly generate a browsable version of the bibliography they want to publish on the Web. Particular attention has been paid to the presentation of bibliographies as hypertext [26]. This not only means that the bibliography is as heavily intralinked as possible, but also that — whenever possible — links are being provided to outside services and resources such as search engines, DOI resolvers, and OpenURL services, thus embedding the bibliography information into the service landscape provided by the Web.

Because import and export are implemented as operations in the user interface, they can only be used interactively. For providing access to the system's information to other applications, a publishing feature has been implemented, which uses much of the export functionality, but provides access to the results through an HTTP Web Service. Users can create and manage publishing channels through a 3-step process (select bibliography, select search criteria, select export format and options), and the publishing channel content can then be retrieved by any application accessing the channel's URI.

5.3 Integration

Even though the system's design is based on the ideas presented here, there still needs some work to be done for integrating a new system into the existing landscape of systems and services inside the university. The integration with two important existing systems thus served as a test case for how easily integration tasks could be accomplished:

- *Web Content Management System (WCMS):* The university supports a central system for managing Web content, and an increasing number of departments and institutes are switching over to this system instead of running their own Web server. The general idea is not to stipulate that everybody must use this system, but to convince potential users that it is in their own interest to do so. Providing an easy and automated mechanism to publish publication lists is one piece in this overall picture.

The WCMS uses an internal format for representing data which is XML-based, and also can be imported into the system. Thus, ShaRef was extended with an export filter for the WCMS, which enables ShaRef users to directly produce Web content from their bibliography. For automatic inclusion of this information, however, the Web Content Management System would have to use a publishing channel providing this XML, so that it could automatically use the most recent publication data. Unfortunately, the WCMS does not support HTTP requests in its code, but this functionality will be added to the system. This will not only allow the WCMS to integrate ShaRef publishing channels, it will generally turn the WCMS into a more versatile component of the evolving IT landscape.

- *University Publication Database:* Another system which exists already and manages related data is a university-wide database of publications. This database has been intended to serve as a source for the yearly research report of the university, and only contains publications with university members as contributors. These publications often are part of the authors' bibliographies anyway, so instead of re-entering this information through the publication database's Web interface, it would be easier to transfer the data from ShaRef to the publication database.

  In terms of integration, this requires that users or organizational units first decide which publication records are relevant, and then initiate the transfer. For a long time, the publication database did not support any import format and only had a form-based Web interface. Recently, a BibTeX import feature has been added, so theoretically the data transfer could be done using BibTeX as the intermediary format. The mid-term goal is to better support the transfer process by directly supporting the publication database with a dedicated XML export format, but as long as there is no XML-based import format and feature, this cannot be implemented.

These examples show that a system designed to serve as a building block in an evolving knowledge organization landscape can be integrated more easily with existing systems. However, in many cases some effort on both sides is still required to accomplish the integration, and this effort is probably hard to avoid. The more important issue is to keep this manual effort as small as possible, so that integration tasks can be accomplished as easy as possible.

Since ShaRef is entirely XML-based, it is also possible to store XML information inside ShaRef which is then re-exported when generating XML as export or publishing format. This way, bibliographic information can be enriched with additional information which is passed through the application and can be used by other applications down the processing pipeline. This makes ShaRef a more versatile component inside Knowledge Organization Mashups, because it simply forwards information which is not part of the built-in data model, rather than ignoring it.

6 Future Work

We have presented a proposal and outlook for constructing knowledge organization systems in a new way, following the general idea of mashups being built from independently constructed components. One of the most important aspects of mashups is that they are built from easily understandable components, and should present themselves as easily understandable components. Because of this, we propose to build these components based on standard XML technologies rather than more advanced technologies for which tools and know-how are not very wide-spread. This clearly is a trade-off between the ease-of-use for developers and the question of how advanced the recommended technologies are.

While the approach presented here is a modest one in terms of advanced underlying technologies, it may have more practical impact because it helps people to help themselves. However, some support is required, and future work in this direction should include methods and tools for tracking and documenting the development of Knowledge Organization Mashups within an organization, and the question how to distribute this information.

The guidelines about concrete technologies (now recommending XML and mentioning RDF only as an additional way to represent information) should be updated according to the development of technologies and tools. However, the acceptance and availability of tools and competent people for XML-based technologies will be much higher than for Semantic We technologies for some time to come, and as long as there is a substantial gap between these two different areas, the simpler and more accepted technologies should be preferred.

7 Conclusions

The current Web trend of *mashups*, simple components which are built on top of other simple components, leads to a dynamism of service development which would not be possible with more complex services or less developer-friendly underlying technologies. We propose to adapt this approach for the construction of *Knowledge Organization Mashups*. Rather than regarding knowledge organization systems and services as standalone applications, they should be regarded as components in an evolving network of service providers and users. This has consequences for the way how knowledge organization applications should be designed. In particular, the data model, format, and interfaces should be designed in a way which facilitates reuse, and the focus should be on designing components which can be easily understood and integrated.

The syntax-based approach of XML technologies proposed for the development of these components is a challenge for component designers to design the syntax as user-friendly as possible. RDF and other Semantic Web technologies would provide a more semantics-oriented foundation, but they also constitute a higher barrier-to-entry than standard XML technologies. As long as average application developers are much more confident and knowledgable in working with XML technologies and tools, these should be supported as the preferred foundation for building knowledge organization mashups.

The approach presented here is a light-weight approach in the sense that it trades semantic expressiveness for ease-of-use and increasing the chances of a dynamic evolution of knowledge organization systems and services within an IT landscape. In a federated environment, this approach is more likely to produce an environment in which information and knowledge can be easily used, shared, and reused and are thus put to their best use.

References

[1] American National Standards Institute. The OpenURL Framework for Context-Sensitive Services. ANSI/NISO Z39.88-2004, April 2005.

[2] Tim Berners-Lee, James A. Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[3] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-2-20041028, October 2004.

[4] Scott Boag, Donald D. Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Candidate Recommendation CR-xquery-20051103, November 2005.

[5] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). World Wide Web Consortium, Recommendation REC-xml-20040204, February 2004.

[6] Dan Brickley and Ramanathan V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. World Wide Web Consortium, Recommendation REC-rdf-schema-20040210, February 2004.

[7] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. World Wide Web Consortium, Candidate Recommendation CR-wsdl20-20060106, January 2006.

[8] James Clark. Associating Style Sheets with XML Documents. World Wide Web Consortium, Recommendation REC-xml-stylesheet-19990629, June 1999.

[9] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. UDDI Version 3.0.2. Organization for the Advancement of Structured Information Standards, UDDI Spec Technical Committee Draft, October 2004.

[10] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. Internet proposed standard RFC 2616, June 1999.

[11] Ned Freed and Nathaniel S. Borenstein. Multipurpose Internet Mail Extensions (MIME) — Part One: Format of Internet Message Bodies. Internet draft standard RFC 2045, November 1996.

[12] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. World Wide Web Consortium, Recommendation REC-soap12-part1-20030624, June 2003.

[13] Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, Volume One. World Wide Web Consortium, Recommendation REC-webarch-20041215, December 2004.

[14] Michael Kay. XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium, Candidate Recommendation CR-xslt20-20051103, November 2005.

[15] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210, February 2004.

[16] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Thomas Nicol, Jonathan Robie, Mike Champion, and Steven Byrne. Document Object Model (DOM) Level 3 Core Specification. World Wide Web Consortium, Recommendation REC-DOM-Level-3-Core-20040407, April 2004.

[17] Mark Nottingham and Robert Sayre. The Atom Syndication Format. Internet proposed standard RFC 4287, December 2005.

[18] Brian M. Oki, Manfred Pflügl, Alex Siegel, and Dale Skeen. The Information Bus: An Architecture for Extensible Distributed Systems. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 58–68, Asheville, North Carolina, December 1993.

[19] Brian J. Peterson, William A. Andersen, and Joshua Engel. Knowledge Bus: Generating Application-focused Databases from Large Ontologies. In Alexander Borgida, Vinay K. Chaudhri, and Martin Staudt, editors, *Proceedings of the 5th International Workshop on Knowledge Representation Meets Databases*, pages 2.1–2.10, Seattle, Washington, May 1998.

[20] Eric Prud'Hommeaux and Andy Seaborne. SPARQL Query Language for RDF. World Wide Web Consortium, Working Draft WD-rdf-sparql-query-20060220, February 2006.

[21] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL Web Ontology Language Guide. World Wide Web Consortium, Recommendation REC-owl-guide-20040210, February 2004.

[22] C. Michael Sperberg-McQueen and Eric Miller. On mapping from colloquial XML to RDF using XSLT. In *Proceedings of 2004 Extreme Markup Languages Conference*, Montréal, Canada, August 2004.

[23] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, October 2004.

[24] Erik Wilde, Sai Anand, Thierry Bücheler, Nick Nabholz, and Petra Zimmermann. Bibliographies as Shared Resources. In *Proceedings of the 2006 IADIS International Conference on Web Based Communities*, San Sebastian, Spain, February 2006.

[25] Erik Wilde. Describing Namespaces with GRDDL. In *Poster Proceedings of the Fourteenth International World Wide Web Conference*, pages 1002–1003, Chiba, Japan, May 2005. ACM Press.

[26] Erik Wilde. Shared Bibliographies as Hypertext. Technical Report TIK Report No. 224, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, Switzerland, May 2005.

[27] Erik Wilde. Towards Conceptual Modeling for XML. In Rainer Eckstein and Robert Tolksdorf, editors, *Proceedings of Berliner XML Tage 2005*, pages 213–224, Berlin, Germany, September 2005.

[28] Dave Winer. RSS 2.0 Specification, July 2003.