

Component-Based Content Linking Beyond the Application

Johannes Meinecke¹, Frederic Majer¹, Martin Gaedke²

¹University of Karlsruhe, Institute of Telematics,
Engesserstr. 4, 76128 Karlsruhe, Germany

{Meinecke,majer}@tm.uni-karlsruhe.de

²Chemnitz University of Technology, Faculty of Computer Science
Straße der Nationen 62, 09111 Chemnitz, Germany
gaedke@cs.tu-chemnitz.de

Abstract. The content of many innovative Web sites today often originates from beyond the application. This paper is concerned with building Web applications that heavily integrate and link content from external sources, like e.g. Web services or RSS feeds. Unlike conventional applications, they are characterized by a very dynamic and distributed information space. In this context, traditional Web Engineering approaches suffer from the fact that they rely too much on a-priori knowledge of existing content structures. We present a support system and a method for building such applications in a very flexible way. Flexibility is achieved by managing links separately from the content in a dedicated Web service and by composing the application from fine-grained, reusable components that realize navigation, presentation, and interaction for the linked content.

Keywords: Web Engineering, Content Linking, Web Services, Reuse, Triple Stores

1 Introduction

Today, Web Engineering is challenged by the construction of a wide variety of Web application types. As one important trend, sites are evolving from isolated content providers to functionality providers that are connected to other parts of the Web beyond simple HTML linking. In particular, Web applications combine their functionality with external services and the content contributed by large communities of participating Web users [11]. Prominent examples include sites like Google Maps, Flickr or del.icio.us that, in addition to offering a visible user interface, also provide programmable interfaces in the form of Web services to enable third parties to build applications on top of them (also referred to as *mash-ups*).

However, the existence of standardized Web service technologies only solves a part of the problem. Beyond just integrating content as e.g. separate sets of resources, real added value is only achieved, when resources from different sources are linked to each other. Here, *link* is understood as the representation of a semantic connection be-

tween two resources, i.e. not exclusively related to navigation. We are especially interested in the question of how to account for this linked content when developing Web applications. In particular, we are addressing the following three challenges:

- **Continuous extensions with new content sources:** Unlike content sources that are under the control of the application provider, the set of autonomous sources that are potentially relevant changes frequently, as old services become obsolete and new services become popular. Hence, the development process must be capable of accounting for originally unknown sources, which are integrated and linked to the existing content later. Ideally, such changes should not require re-implementing or re-generating the system at code level, in order to achieve short revision cycles.
- **A repetitive implementation effort for content source linking:** Web developers perceive *getting content from others* as one of the most frequent problems in Web development work [13]. Concrete problems include distribution, caching, support for multiple service interfaces, or the realization of navigation across the linked content. On the other hand, a considerable part of this functionality does not depend on the application domain, but is rather general in nature. This raises the question of how we can package generic functionality in reusable components that abstract from specific applications.
- **Content sources that are unprepared to be linked.** As they originate from different organizations, they were most likely developed independently from each other. Without the means to make changes to the external sources, the linking structure has to be imposed retrospectively. Similarly, little can be assumed in terms of service capabilities. The interfaces may vary and only provide for basic data retrieval operations, instead of e.g. semantic query features.

In this paper, we present a support system and a method that aims at flexible solutions for these challenges by assembling applications from reusable components and services. In section 2, we introduce an example scenario to clarify the scope of our work. Section 3 presents the Linkbase method together with its individual activities and the architectural outline of the resulting applications. We then describe the implemented 5 contains a brief overview of related work. We conclude with a summary and an outline of planned future work in section 6.

2 The Tourism Portal Scenario

To characterize the targeted type of dynamic Web application, we begin by introducing a fictitious scenario. The scenario is concerned with the development of an advanced Web portal for tourists to aid them in planning their trips in advance. For this purpose, the portal provides reports of other people, who have already visited the travel destinations, as well as all sorts of additional tourist-relevant information. As the planning of trips generally requires accurate and up-to-date information, which the site provider organization cannot gather itself, the envisioned application is supposed to rely on the integration of Web services and other resources offered by third parties. Examples of relevant content include public geographical information about places,

locally relevant news from multiple providers, weather forecasts, and events from multiple calendars. As an example for content that is generated and stored at the site itself, the user community contributes their travel experience in the form of reviews or travelogues. Furthermore, information about relationships between the users of the portal, acquired from the service of a networking portal, can be exploited. The last case of content integration is based on an agreement of the tourism and the networking portal to share services and accounts, and thus allow their respective users to log on to each other's sites through single-sign-on mechanisms.

Fig. 1 gives a simplified overview of the possible content structure inside the portal, as the provider might initially plan it. As indicated, most content originates from outside the portal. In order to make the content more useful, the portal needs some additional knowledge about how the individual resources relate to each other. For example, descriptions about places can be linked to most other resources, as to specify where a given news article is relevant or where a given event occurs. Users are related to reports they authored, photos they took, events they participated etc. Furthermore, there are links that apply to resources of only one source, as e.g. a geographical hierarchy on the places, or the relations among the users in their networking communities.

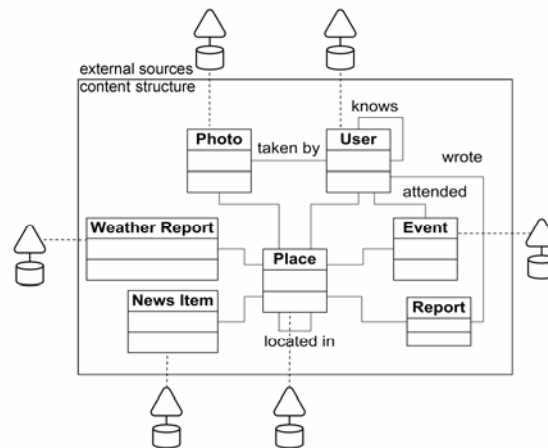


Fig. 1. Content in the Tourism Portal Scenario

Based on the linked content, the tourism portal can offer functionality beyond just letting users browse and contribute reports. For example, users cannot only find out, what other people say about a certain place, but also, whom of their friends they can ask for personal travel experiences, or where their friends will travel within the next year. The benefits produced by the portal can in turn be made available as services to other applications.

3 The Linkbase Method

This section presents the Linkbase method for building applications by linking autonomous content sources with the help of a support system. Fig. 2 gives an overview of

the implementation architecture and the steps to be performed. The central idea is to manage links between arbitrary resources with the help of a central Web service (the Linkbase service) in a uniform way. Inspired by fundamental principles from the Semantic Web, links are seen as triples, i.e. labeled connections between two resources. A resource is anything that can be described with a Uniform Resource Identifier (URI), including conventional Web resources accessible via HTTP and resources queried from Web services. The Web application itself is composed of generic, reusable components that operate on the links from the Linkbase and the content from the different sources to provide navigation, presentation, and interaction to the user.

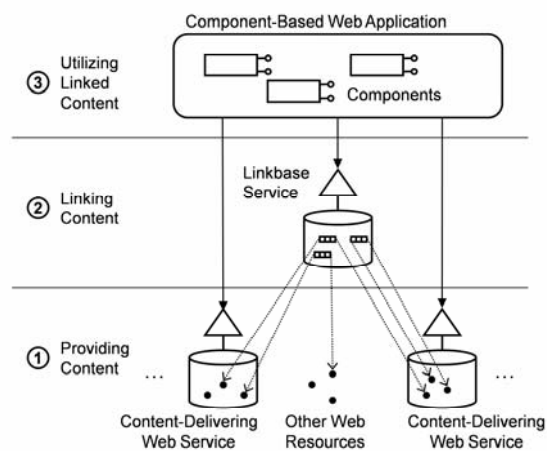


Fig. 2. Architectural Overview of the Linkbase Method

In the following subsections, we give more specific definitions of the concepts in Fig. 2 and discuss three groups of activities that are necessary for building applications. The information space has to be prepared for integration, the Linkbase has to be established, and the Web application has to be assembled. These guidances should not be seen as one-time activities, but more as tasks that become necessary every time the application is extended. Rather than re-implementing the system in every cycle, the Linkbase method focuses on reconstructing only the necessary parts (services and components), as well as their integration with the remaining system at runtime.

3.1 Providing the Content Sources

The aim of the first step is to fulfill the preconditions for the content sources to be linked to each other effectively by making them available in a uniform way. In the case of conventional Web resources¹, the existing Web standards already provide such uniform access methods, as e.g. addressing images via URLs over HTTP. Alternatively, Web services offer more advanced capabilities, e.g. related to querying for

¹ In the sense of RFC 3986.

specific resources. In this paper, we use the term (content-delivering) *Web service* for services that provide access to content to other services and applications via standardized Web protocols, as e.g. SOAP over HTTP or REST.

Along with the more advanced capabilities of Web services as content sources comes a wider choice of access methods, particularly in terms of Web service interfaces. As this variety stands in the way of an effective reuse of content, we propose to use a uniform interface that abstracts from the particular type of content delivered by the service. If the application provider organization controls the service, it can implement the interface directly. Otherwise, the provider creates a wrapping Web service, which is accessed with the uniform interface, and which in turn delegates the content requests to the proprietary interfaces of the third-party services. As one alternative, we propose the specification of an interface in accordance to the CRUDS metaphor [8]. CRUDS allows the querying and manipulation of arbitrary sets of content objects through the operations *Create*, *Read*, *Update*, *Delete* and *Search*. In cases where there is just read-only access, it is sufficient to implement the Read and Search operations. The interface is independent of the content type, as the method signatures contain XML parameters for passing over content objects. Thus, this service provides uniform access to e.g. databases, legacy systems, or, like in our case, other third-party services.

As a second precondition for making the service a part of the information space (in which resources can be linked to each other), we require a method for addressing the supplied resources uniformly. The object identifiers used in the CRUDS interface are inappropriate, as they are not guaranteed to be globally unique. In order to overcome this disadvantage, we propose their extension to so-called *Information Space Identifiers* (ISID). ISIDs are special types of Uniform Resource Nominators (URN) that contain, in addition to the local object identifiers of the resource, also an identifier of the service supplying the resource. The general format of an ISID is:

```
urn:isid:[service id]:[object id]
```

The service identifiers should correspond to the same identifiers that are already used to refer to the service within registries (like e.g. UDDI). Consequently, an application can resolve an ISID by first querying the service's URL from a registry (if not already known) and then invoking its *Read* method with the contained object identifier as a parameter.

Referring to the example in section 2, the first step of the Linkbase method would be to develop a new CRUDS service for handling travel reports, and to create wrapper services for the remaining content sources (weather report, events, etc.). If photos are integrated without any metadata (as e.g. provided by a Web service of a photo sharing site), they can be addressed by URLs, requiring no further implementations. In the other cases, the wrapping Web services introduce new identifiers. For example, an event could be identified with:

```
urn:isid:0a816d35-05fc-41eb-8a76-d54c111c8d2f:20070508-001
```

Here, *0a816d35-...* identifies the wrapping calendar service, and *20070508-001* is the identifier by which the event is known at the external service.

3.2 Linking the Content

The second step of the Linkbase method addresses the actual linking of the resources, within the unified information space. For this purpose, we propose using a *Linkbase service*, which we define as a Web service that provides at least read- and write-access to a set of links, where each link must at least contain a triple of URIs. Each triple consists of a subject, a predicate, and an object. In our case, the subject and object URIs refer to resources provided by the autonomous data sources, and the predicate URI serves as a label for the type of relationship. For the realization of the Linkbase service, there already exists a wide variety of triple stores, many of them related to Semantic Web projects [1], that implement the functionality for storing and managing triples.

Generally, triple stores do not only support storage and retrieval, but also the computation of new triples that logically follow from others. Based on an ontology that captures the necessary knowledge about the types of resources and relationships, the store can infer new facts from old facts. The Linkbase method focuses not so much on storing facts between arbitrary (virtual) concepts, but rather on storing links between concrete resources on the Web. However, reasoning can still be valuable in relieving the application from the burden of computing links by itself. For example, we can declare the *located in* relationship from Fig. 1 as transitive, to better reflect the geographical hierarchy.

So far, we have been concerned with content originating from external sources. In addition to that, it may also be necessary to retrieve the information about which resource is linked to which from the outside. We propose an extension of the triple store concept to not only provide stored and inferred triples, but also triples extracted from external sources at runtime (cf. Fig. 3). When such an extended store receives a query for links, it identifies an appropriate source (like a Web service), queries that source, and returns the result as a set of triples. The idea resembles in some ways the concept of distributed triple stores, where the triple space spreads across several connected stores. However, in this case, the external sources are not restricted to triple stores, but can e.g. also include legacy systems or third-party Web services.

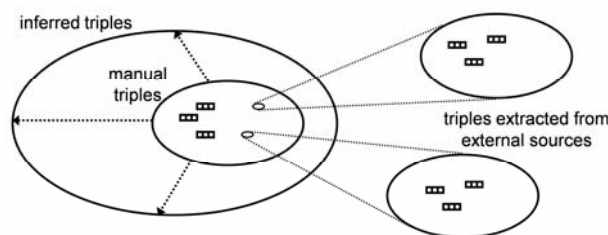


Fig. 3. Inferred and Extracted Triple Space Extensions

With the mentioned extension, there exist three principal ways of inserting links:

- Links are manually created from inside the application itself, e.g. according to information given by a large community of users.
- Links are automatically extracted from external sources at runtime. This is especially applicable when the set of links is very large or changes frequently.

- Links are imported from external sources once, e.g. by applying batch tools. This is preferable in cases where the source's availability is unreliable, or where the procedure of extracting the links takes too long to be performed at runtime.

In the tourism portal scenario, the step *linking the content* comprises setting up one Linkbase service that handles instances of all relationships from Fig. 1. For example, the *taken by* relationship could result in triples like:

```
<http://photo.site/photos/P1010023.JPG>  
<http://purl.org/dc/elements/1.1/creator>  
<urn:isid:77ff72af-...:smithj>.
```

Here, the triple links the URL of a photo with a description of a person supplied by a Web service. Assuming that the photo sharing site supplies this information, a plug-in can be developed for the Linkbase service that extends the triple space with always up-to-date information about the photos that people took. In contrast, the *located in* relationship is relatively stable and can therefore be supplied by a singular import step. The *wrote* relationship is an example for triples created by the application itself, e.g. every time a user contributes a new report.

3.3 Using Linked Content in the Application

Within the third step, the goal lies in allowing developers to construct the (frontend) Web application without having to program it. To achieve this, the Linkbase method focuses *Component-based Web applications*, which we understand as dynamic Web sites that are composed of server-sided, reusable components, usually assembled with the help of a framework. An example for such a framework is our previous work, the WebComposition Service Linking System (WSLS) approach [4], which we adapt to the challenge of linking content from autonomous sources. The idea behind WSLS is to provide a runtime environment for visual, interactive components. At runtime, WSLS allows application developers and administrators to place, rearrange and configure components on pages without recompiling the application. In order to support separation of concerns, components are developed as fine-grained implementation artifacts that can be combined with each other by following the Decorator software design pattern [5].

For the Linkbase method, we supplemented the WSLS approach with a catalogue of components that are dedicated to dealing with linked content. Since both links and content sources are provided in a uniform way, we can restrict the number of components to be implemented and focus on generic functionality. In the following, we present three components from our catalogue to exemplify the support for the three aspects navigation, presentation, and interaction: the Fisheye, the Timeline, and the Content Connector. To be of practical use, they have to be complemented with additional components, like the ones described in [4]. This includes particularly a component that retrieves and caches the content objects from the Web services and that supports the Web service interface chosen for unification (e.g. the CRUDS interface).

Fisheye Component (Aspect Navigation): The Fisheye allows users to navigate through the graph formed by the Linkbase, along selected types of links. This is

achieved by decorating the presentation of a currently active object with smaller navigatable preview presentations of related objects around it (cf. Fig. 4). The name *Fisheye* relates to the impression that the view skims over a web of objects, where the object in focus is always magnified. Technically, the component queries the Linkbase for any resources linked to the currently active resource. When a user activates a hyperlink, the Fisheye component notifies the content-supplying component of the next object to focus.

In order to customize the component for a concrete use case, the configuration has to specify, over which types of links the navigation should occur. In general, it is more advisable to restrict the navigation on a subgraph formed by certain types of connections. Depending on the types, we have to configure, where a related object should be placed (underneath, above, to the left, to the right ...) and how it should be rendered (e.g. with a template).

Example: Given a Linkbase that contains a geographical hierarchy, the Fisheye can be configured to display hyperlinks to geographical places (continents, countries, regions, cities) above, underneath or next to the currently selected place X, depending on whether the places contain, are contained by, or are situated near place X. A similar navigation support can be provided to browse through a network of friends.

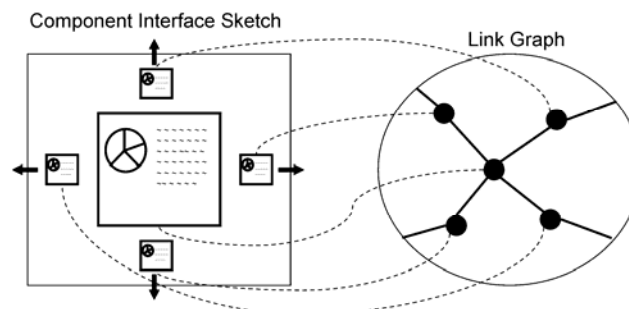


Fig. 4. Fisheye Component

Timeline Component (Aspect Presentation): The Timeline visualizes objects related to a given context in time. To this end, it decorates the presentation of a central object with a time axis and bars that represent the related objects and the time span of the relation (cf. Fig. 5). Similar to the Fisheye, the Timeline queries the Linkbase for all links connected to a particular resource, and renders small presentations of the related objects next to the time bars. The time spans are retrieved either directly from the Linkbase (if the Linkbase supports time-based links [6]) or from common metadata attributes contained in the objects that are supplied by the CRUDS service.

Besides presentation aspects of the timeline axis and bars, the configuration has to specify, how to render the related objects and how to retrieve the time spans.

Example: An application of the Timeline in the running example is to give an overview of submitted travel reports, that relate to a specific travel destination. Alternatively, another interesting view would be to display selected photos and travel reports related to the user or any friend of a user, in order to provide a personalized travel history line.

Content Connector Component (Aspect Interaction): The Content Connector facilitates the interaction between the user and the links by allowing the user to add new links with a single mouse click (cf. Fig. 5). Again acting as a decorator, the component adds hyperlinks or buttons to the presentation of content objects. When activated, it creates a new link in the Linkbase between the decorated object and another object, according to context and configuration. This other object can e.g. be the account of the logged-in user or a pre-defined URI. Alternatively, the user selects an object within a second step, from a choice of objects queried from a CRUDS service.

The configuration determines how the rather technical act of adding a link is presented to the user. Furthermore, it must specify the type of link to add (i.e. the triple predicate URI) and what to link to (see above).

Example: Applied to a list of travel destinations, the Content Connector can provide “Been There”-Buttons, that allow portal users to mark the places they have visited and the tourist activities they have attended on the fly.

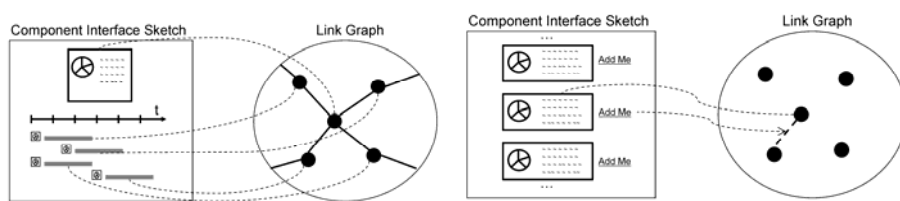


Fig. 5. Timeline Component and Content Connector Component

Based on the described component-based Web application architecture, the third step of the Linkbase method comprises a number of activities to be conducted that follow the principle configuring instead of programming. First, the components required for the construction or extension of the application are provided, either by developing them or by falling back on existing components (e.g. from a component repository). Components and content sources are then registered at the framework, and thus become potential building blocks for the application. Following that, the visible part of the Web application is created or extended by configuring new pages and page sections where the components are instantiated. In other words, the components become responsible for specific parts of the application’s hyperspace. Finally, the generic components are configured in accordance to their intended purpose within these sections. This includes as a vital step the wiring of the component to the registered content sources. Another example of an aspect to be configured is the specification of templates for rendering content objects. Apart from the development of new components, all activities involved in assembling the application are supported by the framework’s Web interface.

4 The Linkbase Applied

The Linkbase method relies on support by system to guide the application developer. In the following section, we present the support system we implemented to validate our approach, including a triple store that realizes the extensions discussed in section

3.2. The Linkbase support system has been used to realize a number of scenarios that demonstrate the applicability of its standard building blocks in different situations.

4.1 Experiments with an Implemented Support System

The central component of the Linkbase support system is the Linkbase service. Rather than using an existing triple store, we implemented a Linkbase service especially with dedicated support for our purposes. The main reason for this was to be able to dynamically extend the triple space with triples extracted from external sources, which is not supported by existing triple stores, and to experiment with extensions to the triple schema (like e.g. time-based triples).

Fig. 6 displays the architecture of the implemented Linkbase that was developed as an ASP.NET Web service. To the outside, the service provides a CRUDS interface, which applications can use for querying (Read, Search) and modifying (Create, Update, Delete) the triple space. Hence, the same service interface can be used to access both the content objects and the links between the objects, resulting in additional re-usability. To provide rudimentary support for reasoning, an inference module was included that computes and stores implied triples every time the triple space is changed. Thus, the Linkbase takes into account properties of link types, which can be defined in an OWL-XML ontology file. For example, the following entry has the effect that whenever a person is linked to an acquainted person, a link is automatically generated in the opposite direction:

```
<owlx:ObjectProperty  
owlx:name="http://xmlns.com/foaf/0.1/knows  
owlx:symmetric="true" />
```

This declaration of link types and knowledge about the linked domain is optional, i.e. links with new types can be added to the Linkbase anytime. Additional triples are generated by the plug-in module, which delegates queries for selected link types to external sources. As the mechanism for extracting external links may vary from source to source (e.g. due to different Web service interfaces), it is performed by plug-ins that are added to the Linkbase at runtime.

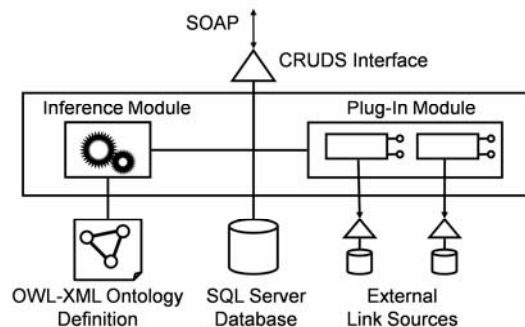


Fig. 6. Linkbase Service Implementation Architecture

To validate the applicability of the support system, we used its various components for the construction of a Web application. This development was conducted in the context of the project Software Engineering for Information Appliances at Home, whose outcome included a Web portal targeted at families at home. Relating to the step *providing the content sources*, we developed a number of CRUDS Web services. Some of them were generated with the help of a custom Visual Studio plug-in, as e.g. a service for handling personal profiles of family members. Others were programmed to wrap existing non-Web systems, as e.g. a calendar service that provides read- and write access to appointments from a Microsoft Exchange server. For example, we created a CRUDS Web service that provides slides as individually addressable content objects (JPG images and metadata entries). Then, we filled it with slides from existing PowerPoint presentation files, including a lecture of 800 slides.

Following that, we set up the Linkbase service to provide the links according to the three principal ways described in section 3.2. For instance, the links that make up the relationships between the family members are specified by the portal users themselves during the lifetime of the Web site. Here, we took advantage of the built-in reasoning support, which in this case supplements the user's input to work out any implied relationships. As an example for a dynamically extracted link type, we developed a plug-in that queries a public Web service of the popular photo sharing site Flickr, to provide links between photos and concepts represented by tags. In the case of the slide service, the hierarchical structure of the presentations (chapters, sections, learning units...) was converted to links during the initial import step.

The visible part of the Web application was exclusively assembled and configured from components, i.e. no code was written to wire the components. Fig. 7 contains a screenshot of the application in administration mode, together with an extract from the configuration screen. The depicted page section is composed of a combination of four components: a CRUDS component to communicate with the profile Web service, a template presentation component to render the personal profile in HTML, a navigation component to provide previous- and next-buttons, and finally, a timeline component, to decorate the person with an overview of related photos and events.

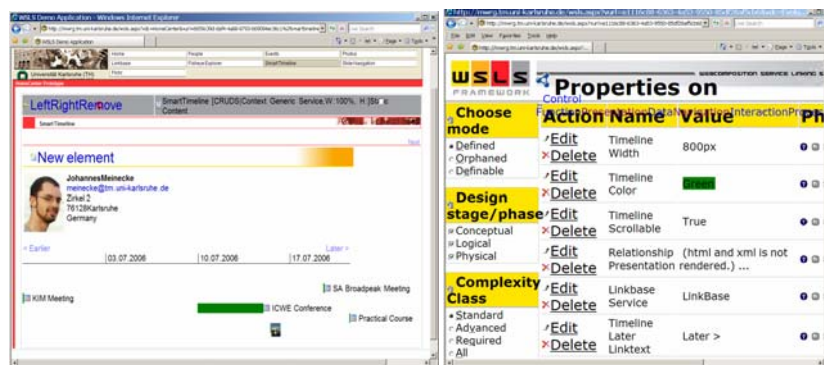


Fig. 7. Timeline Component: Events and Photos

Fig. 8 shows the Fisheye component applied for two different purposes. In the first screenshot, it supports the user in navigating through a family tree and browsing the

profiles of family members. According to the specific configuration, the Fisheye places links to parents above, links to children below and links to partners next to the current profile. In the second screenshot, the same concept is applied to realize a presentation slide browser. Here, the horizontal dimension lets users skim through slides on the same level, while the vertical dimension lets them move up and down the lecture hierarchy, e.g. in order to return to the title slide of the current section.

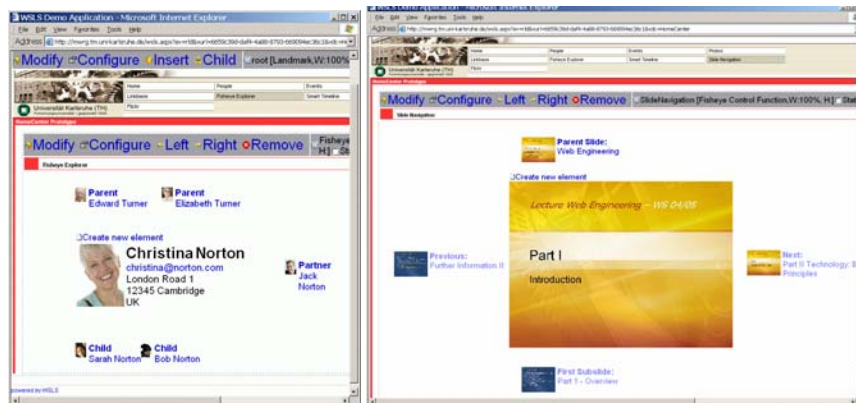


Fig. 8. Fisheye Component: Family Tree and Lecture Slides

All involved content sources were initially unprepared for linking. Some of them were even beyond our control in a sense that we could not influence their content structure (like of the Exchange Calendar or the Flickr Web service). Although programming was necessary, the implementation was limited to generic, application-independent components that can be reused later on to save effort in the long run. Rather than planning everything from the very beginning, we extended the application in several cycles: The first version only featured family members and photos; then the calendar was added, and later on the presentation slides. Hence, the chosen scenario corresponds to the three issues in content linking identified in the introduction.

4.2 Lessons Learned

While the project's outcome confirmed the approach's suitability to meet the targeted problems, we also gained experience related to implementation issues and potential for improvement. Concerning the data model of the link, applications can benefit from extensions of the basic triple concept. Additional information, like the already applied integration of time intervals or the inclusion of standard metadata attributes enable more advanced ways of using the links.

To increase performance, redundant information about the linked resources can be stored with the links in cases where the information is unlikely to change. This proved especially helpful, as it drastically reduced the frequency at which components had to query the linked resources. Beyond that, the need for caching at the application to achieve adequate response times became evident. Within the project, this was achieved with the help of the WSL framework, which caches content retrieved from

CRUDS Web services at the granularity of individual content objects. For example, in order to render an object that is linked with l other objects inside a Fisheye or Timeline component without caching, the Linkbase is queried once and the content-supplying service ($l+1$) times (or once, if the triples contain enough information for rendering the previews). While these measurements can ease performance problems caused by the Web service communication overhead, the Linkbase remains a bottleneck in the architecture. As the number of linked sources grows, it may become necessary to distribute the Linkbase (e.g. as proposed in [3]), which was however beyond of the scope of the presented work. As a major advantage, the Linkbase architecture provides for a great degree of flexibility, accounting for changes to the application's information space from the very beginning.

5 Related Work

In the following, we give a brief overview of several approaches that are related to the challenge of linking autonomous content sources in Web application development.

Several model-oriented Web Engineering methods address the aspect of external content and services. For example, WebML, a visual language for the specification of data-intensive Web applications, has been extended for the model-driven development of Web applications that consume web services as data sources [2]. The focus here lies on applications with most of the data stored in a database (i.e. under the control of the provider), whereas the external content only supplements this data. Directly opposed with respect to this aspect, the HERA methodology targets distributed Web-based information systems where data is retrieved from Web sources with semantic query capabilities [16]. While this approach enables the user to perform very intelligent search capabilities over heterogeneous sources, it requires relatively high development effort for ontology integration, even in cases where such queries are not necessary. The OOWS method has been conceptually extended with support for Web Services [14] and content aggregation [15]. Generally speaking, there is a discrepancy between the very systematic, model-based development methodologies and the dynamic nature of the information space exploited by modern Web applications.

From the architectural point of view, the idea of managing content links in a separate service is closely related to Open Hypermedia Systems (OHS) and the concept of the *link service* found there [12]. This relatively old idea has also been applied to Web technologies [3] as well as to Web services in particular [10]. In the context of this field, our approach can be understood as an application of the OHS concept to a unified information space on top of Web services, and as an alternative method for exploiting the links in a reusable manner. Related to our idea of dynamically extending the triple space, [10] proposes the storage of dynamic hypermedia links that contain queries instead of rigid identifiers, in order to realize stable links on a evolving information space. While this approach does not explicitly target Web services, it could be combined with ours by delegating these queries to the (unified) Web services.

With respect to the goal of composing functionality and content from existing Web-based systems, our approach is related to the field of Semantic Web services. As one example, the WSMX system supports the execution and combination of semanti-

cally described Web services [7]. This approach is very powerful, as the system can dynamically chose and combine appropriate services to fulfill goals given by the requestor. However, it is less suitable for dealing with the specific problem of handling and linking sets of resources provided by these services in a uniform way. As an example for a more manual approach to application composition, [9] proposes the end-user-driven definition and wiring of components that wrap existing Web applications. The idea has advantages in cases where there is no Web service interface to build on, but entails the same problems that are common to all Web site wrapping strategies, e.g. in terms of sensitivity to layout changes on the wrapped sites.

6 Conclusion

The contribution of this work was a novel way of constructing Web applications that integrate content from external sources. The development of such applications require very flexible implementation techniques, as the set of content sources to be integrated is often unknown at design time and is subject to changes later on. In this paper, we described a way to support application development with a support system, whose major component is a Web service for providing links between content objects from different sources (the Linkbase service). To apply this system, we first unify the application's information space by introducing a standardized interface for (wrapping) Web services, and a standardized way of addressing content objects. Content objects are linked by triples of URIs, stored by the Linkbase service. For the application architecture, we propose to assemble Web sites from configurable generic components that work with the unified content sources and links.

We are now working on variations of the triple data model, in order to examine the added value gained by augmenting the triple with metadata. Another open question for future research is the applicability of older distribution concepts for link services to the problem at hand to better address performance issues. Furthermore, dynamic aspects of the Linkbase, like change management, remain to be investigated, also taking into account existing approaches.

Demonstration Videos

Demo videos of the described support system are available for download at the MWRG homepage, <http://mwrgrg.tn.uni-karlsruhe.de/downloadcenter/systems/demos>.

Acknowledgements

This material is partially funded by Microsoft Research Cambridge, within the context of the research project 2005-053.

References

1. Beckett, D.: SWAD-Europe Deliverable 10.1: Scalability and Storage: Survey of Free Software/Open Source RDF storage systems -, W3C: http://www.w3.org/2001/sw/Europe/reports/rdf_scalable_storage_report/ (12.10.2006)
2. Brambilla, M., et al.: Model-driven Development of Web Services and Hypertext Applications. in SCI2003. Orlando, Florida (2003)
3. Deroure, D., et al.: A Distributed Hypermedia Link Service. in Third International Workshop on Services in Distributed and Networked Environments: IEEE (1996) 156-161
4. Gaedke, M., Nussbaumer, M., and Meinecke, J.: WSLS: An Agile System Facilitating the Production of Service-Oriented Web Applications, in Engineering Advanced Web Applications, S.C. M. Matera, Editor, Rinton Press (2005) 26-37
5. Gamma, E., et al.: Design patterns: elements of reusable object-oriented software. Addison-Wesley professional computing series, Reading, Mass.: Addison-Wesley (1995) xv, 395
6. Gutierrez, C., Hurtado, C., and Vaisman, A.: Temporal RDF. in European Conference on the Semantic Web (ECSW'05) (2005) 93-107
7. Haller, A., et al.: WSMX-a semantic service-oriented architecture. in 2005 IEEE International Conference on Web Services (ICWS 2005). Orlando, Florida (2005) 321-328
8. Ibm: Elements of Service-Oriented Analysis and Design -, IBM Homepage: <http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/>
9. Ito, K. and Tanaka, Y.: A visual environment for dynamic web application composition. in ACM Conference on Hypertext and Hypermedia. Nottingham, UK ACM (2003) 184-193
10. Karousos, N., et al.: Offering Open Hypermedia Services to the WWW: A Step-by-Step Approach for Developers. in Twelfth International Conference on World Wide Web. Budapest, Hungary (2003) 482-489
11. O'reilly, T.: What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software - Online Article: <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html> (18.10.2005)
12. Pearl, A.: Sun's Link Service: A Protocol for Open Linking. in 2nd Annual ACM Conference on Hypertext. Pittsburgh, USA: ACM Press (1989) 137-146
13. Rosson, M.B., et al.: "Designing for the Web" Revisited: A Survey of Informal and Experienced Web Developers. in 5th International Conference of Web Engineering (ICWE 2005). Sydney, Australia: Springer (2005) 522-532
14. Ruiz, M., et al.: A Model Driven Approach to Design Web Services in a Web Engineering Method. in 1st International Conference on Advanced Information Systems Engineering Forum (CAiSE Forum). Porto, Portugal (2005)
15. Valderas, P., Fons, J., and Pelechano, V.: Extending Navigation Modeling to Support Content Aggregation in Web Sites in Fourth International Conference on Web Engineering (ICWE'04). Munich, Germany: Springer (2004) 98-102
16. Vdovjak, R., Barna, P., and Houben, G.J.: Designing a Federated Multimedia Information System on the Semantic Web. in CAiSE: Springer (2003) 357-373