

TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Informatik XIX

**Facilitating Emergent and Adaptive Information
Structures in Enterprise 2.0 Platforms**

Christian Alexander Neubert

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität
München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. M. Bichler

Prüfer der Dissertation:

1. Univ.-Prof. Dr. F. Matthes
2. Univ.-Prof. Dr. J. Schlichter

Die Dissertation wurde am 29.05.2012 bei der Technischen Universität München
eingereicht und durch die Fakultät für Informatik am 21.09.2012 angenommen.

Zusammenfassung

Enterprise 2.0 bezeichnet den Einsatz von Plattformen für soziale Software in Unternehmen oder zwischen Unternehmen und ihren Partnern oder Kunden. Dazu gehören Wikis, Blogs, Dateifreigaben und soziale Netzwerke. Da häufig nur textuelle Inhalte auf diesen Plattformen verwaltet werden, wird mit steigender Anzahl an Nutzern, die Inhalte beitragen, der effiziente Zugang zu und die Suche nach Informationen erschwert. In neueren Enterprise 2.0 Lösungen können IT-Experten Vorlagen, Tabellen oder semantische Annotationen erstellen, um diese Situation zu verbessern. Dies führt allerdings zu bekannten Kommunikationsproblemen zwischen der IT und dem Geschäft (Missverständnisse, Verzug, Kosten) und einer Diskrepanz zwischen rigiden Informationsstrukturen und dem schnellen und einfachen Erstellen von Inhalten.

Die in dieser Forschungsarbeit entwickelten hybriden Wikis befähigen Nutzer Inhalte (z.B. Wikiseiten, Blogeinträge) und Links mittels Attributen, Typen und Integritätsbedingungen inkrementell zu strukturieren und zu klassifizieren, um diese Herausforderungen zu adressieren. Nutzer interagieren mit diesen Konzepten über eine ihnen vertraute Oberfläche unter der Verwendung von Formularen, Tabellen, Autovervollständigung und attributbasierten Anfragen. Aus diesen Interaktionen entstehen emergente Informationsstrukturen als Teil der täglichen Geschäftsaktivitäten. Bei Bedarf können Informationsstrukturen von Nutzern auch fest definiert werden, ähnlich einem objektorientierten Datenbankschema.

Im Rahmen dieser Arbeit wird ein konzeptuelles Modell zur Unterstützung emergenter Informationsstrukturen vorgestellt, welches orthogonal auf nutzererstellte Inhalte in Enterprise 2.0 Plattformen anwendbar ist. Es werden Methoden präsentiert, die unter der Verwendung von Vorschlägen das Strukturieren von Informationen erleichtern. Des Weiteren werden Mechanismen eingeführt, mit denen strukturelle Abweichungen erkannt und vereinheitlicht werden können. Die Innovativität von hybriden Wikis wird anhand eines Vergleichs mit 28 verwandten Forschungsarbeiten sowie einer Analyse der Funktionen von neun prominenten Enterprise 2.0 Lösungen demonstriert. Ein weiterer Bestandteil dieser Arbeit ist die Implementierung hybrider Wikis basierend auf einer bestehenden, kommerziellen, in Java geschriebenen Enterprise 2.0 Plattform. Zunächst wird die zugrundeliegende Systemarchitektur sowie die Schnittstellen zur Basisplattform erklärt. Anschließend wird das Zusammenwirken von Typen, Attributen und Integritätsbedingungen mit Persistenzschicht, Systementwicklung, dynamischer Zugriffskontrolle, Suche, Link Management, Tagging, Versionshistorie und der Benutzeroberfläche diskutiert.

Zur Validierung wird die Qualität emergenter Informationsstrukturen über die Durchführung eines kontrollierten Laborexperiments bewertet. Der produktive Einsatz hybrider Wikis in sechs Organisationen aus verschiedenen Anwendungsbereichen zeigt, dass Nutzer in Unternehmen tatsächlich ohne Unterstützung von IT-Experten Informationsstrukturen gemäß sich ändernder Anforderungen erstellen und anpassen.

Abstract

Enterprise 2.0 is the use of emergent social-software platforms within companies, or between companies and their partners or customers, including wikis, blogs, file sharing, and social networks. Due to the free-form nature of the content managed on these platforms, efficient information retrieval and access becomes a challenge once a significant number of users contribute content. In more recent Enterprise 2.0 systems, IT experts can define templates, structured tables, or semantic annotations to improve this situation. However, this leads to well-known problems of business-IT communication (misunderstandings, delay, costs) and a mismatch between rigid information structures and fluid content authoring.

To address this issue, we developed the concept of Hybrid Wikis, which empower business users to incrementally structure and classify content objects (e.g., wiki pages, blog posts) and links using attributes, types, and constraints. Business users interact with these concepts using a familiar interface based on forms, spreadsheet-like tables, autocompletion, and attribute-based queries. From these interactions, information structures evolve as part of the users' day-to-day business activities. If required, business users can make these information structures explicit and enforce them like an object-oriented database schema.

We introduce a conceptual model supporting emergent information structures orthogonally applicable to user-generated content in Enterprise 2.0 platforms. Furthermore, we present techniques facilitating information structuring using suggestions and introduce mechanisms to detect and consolidate deviations in structures. The novelty of Hybrid Wikis is shown by a comparison with 28 related research works and an analysis of the functional capabilities of nine prominent Enterprise 2.0 vendor solutions. As part of this thesis, Hybrid Wikis have been implemented based on an existing commercial Enterprise 2.0 platform written in Java. We first explain the underlying system architecture and the interfaces to the base platform. We then discuss how types, attributes, and constraints interact with persistence management, system evolution, dynamic access-control, search mechanisms, link administration, tagging services, version history, and user interface design.

To validate our approach, we present the results of a controlled laboratory experiment evaluating the quality of the emerging information structures compared to UML models. The productive use of Hybrid Wikis in six organizations from different application domains shows that business users indeed create and adapt information structures according to changing demands without the need to involve IT experts.

Acknowledgment

This thesis emerged from my work as a research assistant at the Chair for Informatics 19 (sebis) at the Department of Informatics of the Technische Universität München. At this point, I would like to express my gratitude to all those, who supported me in my work during the last four years.

First, I would like to thank my doctoral advisor, Prof. Dr. Florian Matthes, for providing me the opportunity to work on this interesting research topic and for his suggestions and advice that greatly contributed to the success of this work. I would also like to thank Prof. Dr. Johann Schlichter for his conversations about the subject and for being the second supervisor of my thesis.

My thanks goes to my colleagues for their time for discussions that greatly helped to develop the ideas underlying Hybrid Wikis. I would like to thank Dr. Sabine Buckl and Dr. Christian Schweda from the enterprise architecture management team for being the ‘obstetricians’ of Hybrid Wikis. In particular, I thank Dr. Thomas Büchner and Alexander Steinhoff from the social software team for helping me to bring up Hybrid Wikis, from a clumsy infant to a young, confident adult. I would also like to thank my colleagues Ivan Monahov, Sascha Roth, Alexander Schneider, and Christopher Schulz for the constructive cooperation in teaching and researching as well as for annoying me by constantly submitting bugs. This really helped to improve Hybrid Wikis.

I would like to thank the students, whose thesis I supervised the last four years. Thanks to Jacob Class, Pawel Dacka, Markus Dauberschmidt, Andy Großmann, Matthias Hunecker, Julian Lebherz, Matti Maier, Andreas Mirbeth, Julian Sommerfeldt, Katharina Rau, Thomas Reschenhofer, Claudia Sroke, Manuel Tremmel, Alexej Utz, and Bernhard Waltl. I really enjoyed working with you.

I also want to express my gratitude to my partner, Claudia Beer. Thanks for your love, being so patient, and constantly reminding me that there is life outside of my computer. Finally, I am most grateful to my family. Thanks to my parents Brigitta and Wolfram, and my sister, Kristina Neubert. Without your support, encouragement, and love during my whole life this work would not have been possible.

Garching b. München, 15.05.2012



Christian Neubert

1	Introduction and Overview	1
1.1	Motivation and problem statement	1
1.2	Research questions	3
1.3	Research design	4
1.4	Outline of the thesis	7
2	Towards Hybrid Wikis	9
2.1	Principles guiding the development of Hybrid Wikis	11
2.1.1	Simplicity over expressivity	12
2.1.2	Data over schema	12
2.1.3	Evolution over rigidity	13
2.2	Example scenario	13
3	Hybrid Wikis	17
3.1	Concepts supporting emergent information structures	17
3.1.1	Spaces	18
3.1.2	Content objects	18
3.1.3	Attributes	23
3.1.4	Types	23
3.1.5	Values	24
3.1.6	Structured	30
3.1.7	Attribute definitions	30
3.1.8	Constraints	33
3.2	Emergent structures in Enterprise 2.0 platforms	40
3.2.1	Participation model	40
3.2.2	Evolution of emergent information structures	41
3.2.3	Access control for structuring concepts	44
3.2.4	Integration of structuring concepts in Enterprise 2.0 services	46
3.3	Techniques facilitating information structuring	49
3.3.1	Suggestions	49

3.3.2	Transitions	56
3.3.3	Schema-based information consolidation	66
3.3.4	Data-based schema adaption	68
3.4	Views based on structured information	68
3.4.1	Built-in views	69
3.4.2	Custom views	77
3.4.3	Structured search	79
3.4.4	Advanced UI operations	80
4	Implementing Hybrid Wikis in Tricia	83
4.1	Introduction to Tricia	83
4.1.1	Architecture	84
4.1.2	Data modeling framework	86
4.1.3	Interaction framework	90
4.1.4	Access control framework	90
4.1.5	Plugins and extension points	91
4.1.6	Enterprise 2.0 platform Tricia	91
4.2	Implementing Hybrid Wikis	91
4.2.1	Data model	92
4.2.2	Extended architecture	110
4.2.3	Suggestions	112
4.2.4	Transitions	116
4.2.5	Consolidation	118
4.2.6	Export and import of structured content	119
4.2.7	Collaborative information management on federated data sources	120
5	Related Work	123
5.1	Semantic Web	126
5.2	Wiki templates	128
5.3	DynaTable	131
5.4	DBpedia	132
5.5	Semantic wikis	133
5.5.1	Semantic MediaWiki	134
5.5.2	Semantic Enterprise Wiki (SWM+)	135
5.5.3	AceWiki	136
5.5.4	OntoWiki	137
5.5.5	Kaukolu	139
5.5.6	Artificial Memory	140
5.5.7	HYENA	141
5.5.8	IkeWiki	143
5.5.9	Makna	144
5.5.10	KiWi	145
5.5.11	Rhizome	147
5.5.12	SHAWN	147
5.5.13	RISE	148
5.5.14	SemperWiki	149
5.5.15	SweetWiki	149

5.5.16	SOBOLEO	150
5.5.17	KnowWE	152
5.6	Corporate Semantic Web	152
5.7	Freebase	152
5.8	MoKi	154
5.9	SnoopyDB	155
5.10	Social Infobox	156
5.11	TWiki	157
5.12	DBWiki	158
5.13	Summary and comparison with Hybrid Wikis	159
5.13.1	Semantic annotations	159
5.13.2	Towards simplicity	160
5.13.3	Structures for public use	161
5.13.4	Social tagging	161
5.13.5	Applicability in enterprises	162
6	Application and Evaluation	163
6.1	Applying Hybrid Wikis	163
6.1.1	Wiki4EAM Community	163
6.1.2	InfoAsset AG	173
6.1.3	TU München (sebis)	174
6.1.4	Pixida GmbH	176
6.1.5	Summary	178
6.2	Experimental examination of models created with Hybrid Wikis	179
6.3	Evaluating and visualizing the structural evolution of Hybrid Wikis	182
7	Conclusion	185
7.1	Summary	185
7.2	Outlook	186
7.2.1	Type hierarchies	186
7.2.2	Empirical evaluation of the use of Hybrid Wikis	187
7.2.3	Towards user-adaptive information systems	188
A	Appendix	189
	Bibliography	191

List of Figures

1.1	Cooperative information management activities of persons and groups in an Enterprise 2.0 platform based on unstructured content.	2
1.2	Communication problems can lead to a mismatch between demands and requirements, latency between demand and release, and a cold start after schema migration.	3
1.3	Information systems research framework according to [He04].	5
2.1	Hybrid Wikis combine schema-based structured information with unstructured information provided by Enterprise 2.0 platforms.	10
2.2	Enterprise 2.0 platform extended with structuring concepts (attributes, types, constraints) provided by Hybrid Wikis.	11
2.3	Exemplary information model used within a small company's intranet for project and customer management.	15
3.1	Core concepts provided by Hybrid Wikis.	18
3.2	Access control for content objects.	21
3.3	Access control model according to [BMN10a].	21
3.4	Default access rights derived from a space.	22
3.5	Concepts supporting the structuring of content objects.	25
3.6	String values.	26
3.7	Internal and external link values.	27
3.8	Hypertext values.	28
3.9	Record values.	29
3.10	Structured link values.	30
3.11	Simplified model representing structured content objects.	31
3.12	Core concepts specifying types and integrity constraints.	32
3.13	Simplified model specifying types and integrity constraints.	34
3.14	Applying constraints when displaying a content object.	35
3.15	Alternative validation model using a context for applying integrity rules.	36
3.16	Enumeration constraint.	38

List of Figures

3.17	Constraints for records, internal links, and structured links.	39
3.18	Integrated data-driven information management and schema evolution via col- laboration.	42
3.19	Degrees of the structure's consolidation.	43
3.20	Implicit and explicit schema in Hybrid Wikis.	43
3.21	Changing a value's data type from string to (internal) link.	57
3.22	Transitions from type to attribute (a), tag to attribute (b), and built-in markup to attribute.	59
3.23	Transition supporting the merge of two attributes.	61
3.24	Objectification of structured values.	62
3.25	Objectification of a semi-structured table.	64
3.26	Objectification of unstructured content.	65
3.27	The content view shows unstructured information, links, and tags of a wiki page.	70
3.28	The hybrid view shows unstructured content on the left and structured content on the right.	71
3.29	Autocomplete suggestions for attribute values.	72
3.30	Attributes, types, and built-in content presented in the structured view enlarged to the full size of the webpage.	73
3.31	The structured link view allows to attach attributes and types to hyperlinks.	73
3.32	The type table view shows all instances of a type.	74
3.33	The attributes view shows all attributes of a type.	75
3.34	The types view shows the all available structures within a space.	76
3.35	Visualizing emergent information structures according to [Ut11].	77
3.36	Custom table and list view embedded in the built-in content of a wiki page.	78
3.37	The search interface allows a faceted drill-down based on the structured contents.	79
3.38	Query and view configuration embedded in wiki text.	80
4.1	Architectural overview of a typical web application implemented based on the Tricia platform.	84
4.2	Example data model consisting of wikis and wiki pages and their relations.	86
4.3	Concepts of Tricia's data modeling framework.	87
4.4	MVC pattern realized in Tricia.	90
4.5	Plugins provided by Tricia before Hybrid Wikis.	91
4.6	Data model underlying Hybrid Wikis.	92
4.7	Mixin <i>Structured</i> applied to some content objects.	93
4.8	Assignment of a type by means of tagging services.	94
4.9	The mixin <i>Structured</i> is based on the mixin <i>Taggable</i>	96
4.10	Storing and indexing of key-value pairs.	99
4.11	Link structures are represented by an entity with mixins	100
4.12	Attributes and constraints integrated in the Tricia data model.	102
4.13	Tricia's validation model checking the consistency of entities with properties and roles.	103
4.14	Attributes integrated in the Tricia data model.	104
4.15	Tricia's validation model extended with attributes and constraints.	105
4.16	Interactive management of values with data types.	106
4.17	Formatting values according to data type and user language.	107
4.18	Validation while editing values.	108

4.19	Attribute definition with constraints.	109
4.20	Plugins provided by Tricia including Hybrid Wikis.	110
4.21	Constituents of the structured plugin.	111
4.22	Constituents of the hybrid wiki plugin.	111
4.23	Autocomplete suggestions for attribute keys.	113
4.24	Autocomplete suggestions for attribute values.	113
4.25	Data type suggestions are provided while entering values.	114
4.26	Suggestions and autocomplete support for types.	115
4.27	Improved input support for values based on attribute definitions and constraints.	115
4.28	Facilitating data entry based on attribute definitions and constraints.	116
4.29	Transferring unstructured content to an attribute.	117
4.30	Objectification of a semi-structured table from the built-in content of a wiki page.	118
4.31	Applying changes in the schema to the instances.	119
4.32	An external object (from SharePoint) describing a lecture with unstructured, additional content, external attributes (e.g., Speaker), and locally stored attributes (e.g., Betreuer) which are linking to external objects (from Exchange) representing persons (e.g., Neubert Christian) according to [RW11].	121
5.1	sebis Enterprise 2.0 Tool Survey result matrix 2010 showing some categories, services, and vendor ratings.	124
5.2	An RDF triple represented by subject, predicate, and object according to [Ha04].	126
5.3	Layers of the Semantic Web. Source: http://www.w3.org/2007/03/layerCake-small.png ; visited on February 10th 2012.	127
5.4	Wikipedia template representing Austrian towns applied to Innsbruck according to [AL07].	128
5.5	Wiki edit template specifying two sections by using wiki markup (Bibliographic Data and Summary) with properties (e.g., Author) and placeholder input fields (wikiTextInput) according to [HLS05]	130
5.6	A wiki page using Lightly Constrained Templating mechanisms indicating validation errors caused by deviations from the master template according to [DIVZ08].	131
5.7	Code embedded in a wiki page (lower part of the figure) to render a table (upper part of the figure), named Companies, with data defined in a separate space according to [Ar09].	132
5.8	DBpedia as a hub of interlinked data according to [Bi09].	133
5.9	Semantic annotations used in the markup editor provided by Semantic MediaWiki.	135
5.10	Semantic Toolbar provided by SWM+ showing categories and properties.	136
5.11	Predictive editor in AceWiki using the controlled natural language ACE to create sentences.	137
5.12	Resource editor provided by OntoWiki.	138
5.13	Kaukolu's page editor showing suggestions based on imported ontologies according to [Ki06].	140
5.14	Hierarchical document in Artificial Memory showing the triples of a paper entitled <i>ArtificialMemory Prototype for Personal Semantic Subdocument Knowledge Management</i>	141
5.15	HYENA's Eclipse-based offline RDF editor according to [Ra10b].	142

5.16	HYENA's web editor showing a wiki page selected in the list of available RDF resources according to [Ra10b].	143
5.17	The main window of IkeWiki showing navigation functionality on the left, wiki content in the middle, and meta data (e.g., incoming links) on the right. Source: http://www.wikimatrix.org/screenshots/screen_28_1.png ; visited on February 19th 2012.	144
5.18	Makna assistant to insert predicates in the wiki text according to [DST06].	145
5.19	A semantic wiki page created based on the KiWi platform according to [Sc09].	146
5.20	Editor provided by Rhizome showing semantic annotations in its own markup language (ZML) according to [So05].	147
5.21	Editor provided by SHAWN using colon-separated key-value pairs for information structuring embedded in the page content according to [Au05b].	148
5.22	SemperWiki editor showing keys (e.g., <i>dc:author</i>) and values (e.g., " <i>Explanation</i> ") in the wiki page content according to [Or05].	150
5.23	Editor provided by SweetWiki showing tag suggestions as the user enters keywords according to [Bu11a].	151
5.24	Editor provided by SOBOLEO showing tag suggestions as the user enters keywords.	151
5.25	Editor provided by KnowWE showing knowledge used for problem-solving purposes and recommended solutions according to [BRP11].	153
5.26	A Freebase topic <i>Development Project DISS</i> with attribute <i>Budget</i> typed as <i>Project</i>	154
5.27	MoKi editor showing a wiki page with textual description, hierarchical structure, and properties according to [Gh09].	155
5.28	Editor provided by SnoopyDB supporting structured tags (in key:value format) applied to a Flickr photo (http://www.flickr.com ; visited on February 22nd 2012).	156
5.29	Editor of Social Infobox showing a resource with properties and suggestions.	157
5.30	Editing a template for contacts in TWiki.	158
5.31	Embedded queries and update forms in DBWiki's pages according to [Bu11b].	159
6.1	Cluster map embedded in a wiki page provided by the SyCa-Tool based on models derived from structured pages in Hybrid Wikis	165
6.2	Information model used by the KVB for the management of the EA.	168
6.3	Emergent information model used by UniCredit Business Integrated Solutions S.C.p.A. (formerly UniCredit Global Information Services S.C.p.A.) for the management of infrastructure elements.	170
6.4	Emergent information model used by Miles Group GmbH for the management of invoices.	172
6.5	Emergent information model used by InfoAsset AG for the management of issues.	174
6.6	Emergent information model used by sebis to manage students and exercises of a course.	175
6.7	Emergent information structures used by Pixida GmbH in the human resource department for business process documentation according to [Gr12].	177
6.8	Default values used as placeholders facilitating the data entry of customer addresses according to [Gr12].	178
6.9	Metric used to experimentally evaluate the quality of UML models and models created with Hybrid Wikis according to [GP01, Kr95, Ra11].	181

6.10 Visualizing the structural evolution of a wiki page with attributes, types, and tags according to [Tr12].	183
A.1 The conceptual model of Hybrid Wikis.	190

List of Tables

3.1	Transitions to data types String, Date, Number, and Hypertext.	58
3.2	Transitions to data types Record and ExternalLink.	58
3.3	Transitions to data types InternalLink and StructuredLink.	58
3.4	Transitions from plain text.	60
3.5	Transitions from hypertext.	60
3.6	Transitions from a link.	61
3.7	Table as part of the built-in content describing two projects with their budgets.	64
4.1	Mandatory mixins and their usage in the core plugin and in the wiki plugin.	89
4.2	Internal and external representation of values according to [Ma11].	108
6.1	Hybrid Wikis used for enterprise architecture management.	179
6.2	Hybrid Wikis applied in enterprises supporting customer-relationship-management, human resources, events, meetings, and accounting.	179
6.3	Hybrid Wikis applied in enterprises supporting the management of projects, tasks, configurations, and templates.	180
6.4	Hybrid Wikis applied in enterprises supporting personal information management, licenses, lectures, and publications.	180

1.1. Motivation and problem statement

In order to keep pace with the growing amount of digital information that has to be managed enterprises have to adopt new tools and methods [EM00]. Enterprise 2.0 platforms are increasingly being used as lightweight shared content repositories that allow users to collaboratively gather, manage, and consolidate information that was previously scattered across emails, files on personal computers, and paper documents [Mc09]. “Enterprise 2.0 is the use of emergent social-software platforms within companies, or between companies and their partners or customers” [Mc06], including wikis, blogs, file sharing, and social networks (cf., Figure 1.1).

Having this information integrated in a central place, being able to search it, and to connect related pieces of information with hyperlinks is a major advance. “Enterprise 2.0 technologies have the potential to let an intranet become what the Internet already is: an online platform with a constantly changing structure built by distributed, autonomous and largely self-interested peers. On this platform, authoring creates content; links and tags knit it together; and search, extensions, tags and signals make emergent structures and patterns in the content visible, and help people stay on top of it all.” [Mc06]

Content managed on such platforms is characterized by its free-form nature. For example, wikis consist of plain textual, unstructured, user-generated content [O’05]. This makes it convenient for business users to immediately react to constantly changing demands resulting from the growing dynamics of today’s business: since content is not captured in a rigid schema in these platforms, users can flexibly share and organize it according to their needs. However, due to the unstructured form of such content, efficient information retrieval and access becomes challenging once a significant number of users contribute content. Soon, information needs to be accessible in more structured ways. For example, it is not possible to query the contents for a company’s research projects that started back in 2010 or to export data about these

1. Introduction and Overview

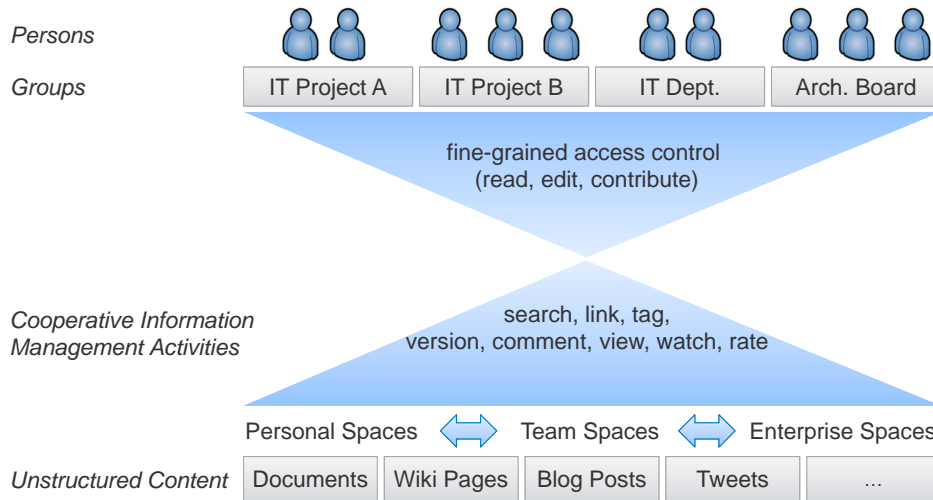


Figure 1.1.: Cooperative information management activities of persons and groups in an Enterprise 2.0 platform based on unstructured content.

projects to a spreadsheet. So even if only rudimentary structured querying functionality is required, enterprises have to resort to separate applications, usually specialized to manage information of particular domains (like employees, projects, or customers) or they have to develop customized solutions. In both cases the advantages of storing information in a central repository are lost.

In more recent Enterprise 2.0 systems, users are enabled to provide structured information, for example by using wiki templates [DIVZ08, HLS05] or table extensions [Ar09] to fill in data. This way, diverging knowledge acquisition is counteracted and information redundancies are avoided. Recently, semantic technologies [Fe11] have been integrated in Enterprise 2.0 applications [Pa11]. For instance, semantic wikis allow to combine textual contents with structured data [KVV06]. Typically, their users have to manually provide data in the form of semantic annotations to wiki pages or parts thereof. The structured part of the information in the wiki can then be queried similar to the contents of a database.

Templates, table enhancements, and semantic extensions can be used to structure arbitrary information. However, they are rarely used as general structuring means that users dynamically adapt to new needs in practice. Instead, structures are often pre-configured and adapted by IT experts in order to solve rather specific problems [Gh09, Ho09, TGP11]. In many cases this leads to well-known problems of business-IT communication as for example when carrying out classical software engineering projects [Bo91] (cf., Figure 1.2):

- Mismatch between demand and realization: The structures delivered do not fulfill the requirements.
- Latency between demand and release: The structures are not available when they are required.
- Cold start after schema migration: No data is provided when the changed structures are available.

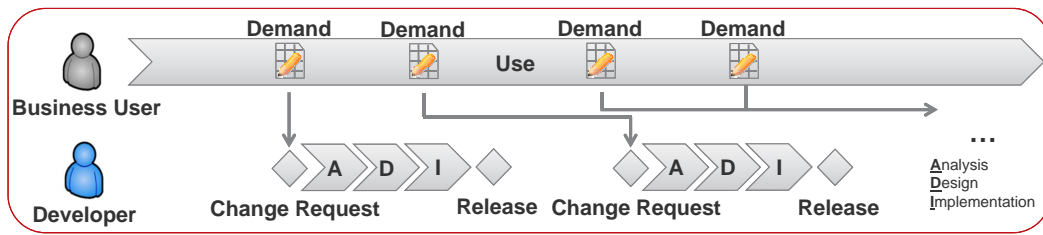


Figure 1.2.: Communication problems can lead to a mismatch between demands and requirements, latency between demand and release, and a cold start after schema migration.

In this thesis, we present a novel approach to mitigate these challenges: Hybrid Wikis.

1.2. Research questions

In order to make our research objective more manageable we split the goals of this thesis in six research questions. These provide guidance to the overall research process and build a foundation for the evaluation of our results. Moreover, answering the answers to our questions add value to an existing knowledge base [GL10].

Research question 1: Is it possible to facilitate emergent and adaptive information structures in Enterprise 2.0 platforms?

This is the overall research question. We break it down into the following subquestions:

Research question 2: Which concepts support information structuring in Enterprise 2.0 platforms?

This question focuses on the identification of concepts necessary to support business users in information structuring. We introduce the relevant structuring concepts in Section 3.1.

Research question 3: Which techniques facilitate information structuring in Enterprise 2.0 platforms?

This question aims at finding a set of mechanisms applicable to the identified structuring concepts (cf., research question 2). In particular, we examine how business users interact with these structuring concepts, how information structuring can be facilitated for business users, and how emergent information structures can be adapted by them. The identified techniques are presented in Section 3.3.

Research question 4: How can information structuring be realized in an Enterprise 2.0 tool?

This question aims at finding an appropriate way of implementing information structuring in Enterprise 2.0 platforms. It focuses on technical aspects in particular, such as an efficient support for structuring information or technical integration of information structuring into

existing Enterprise 2.0 platforms. In Chapter 4, we discuss how information structuring can be supported from a technical point of view.

Research question 5: How do approaches proposed by scientific literature and commercial vendors support information structuring compared to our solution?

This question aims at finding out what the information structuring capabilities of already existing tools and prototypes are, what they have in common with our approach, and in what ways they differ. In Chapter 5, we introduce several approaches and relate them with our solution.

Research question 6: What is the experience gained in applying our solution in practice? Does the application lead to emergent information structures? What quality have structures created with our solution?

These questions aim at the application and evaluation of the developed artifact in practice. In particular, the questions' purpose is to find out which structures emerge and how structures evolve when our solution is applied in business contexts. Furthermore, the questions help to compare structures of our solution with those of other structuring approaches. In Chapter 6 we introduce practical experiences made with our approach. In particular, we present selected industry case studies and results of an experimental examination.

In order to address these six research questions we applied the research methodology as introduced in Section 1.3.

1.3. Research design

The research conducted in this thesis is based on the design science approach. The research questions given in Section 1.2 directly point to the design science approach as introduced by [He04]. The authors of this article describe design science as a problem-solving process based on the principle “that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact”. They point out that “design science addresses research through the building and evaluation of artifacts designed to meet the identified business need”. An overview of the proposed information systems research framework is depicted in Figure 1.3.

This thesis provides an analysis of scientific literature concerned with tools and prototypes supporting information structuring. Furthermore, it presents the results of a survey on prominent open source and commercial Enterprise 2.0 platforms (cf., Chapter 5). These efforts can be considered as “framing research activities to address business needs”. This way, we assured research relevance.

The artifact designed in this thesis consists of a set of concepts and techniques facilitating the evolution and adaption of information structures in Enterprise 2.0 platforms. An instantiation is provided by implementing these concepts and mechanisms based on an existing Enterprise 2.0 platform, resulting in an innovative solution.

The developed solution is applied to several enterprises from the problem domain in order to evaluate our approach and to show its usefulness. Since “truth and utility are inseparable” we

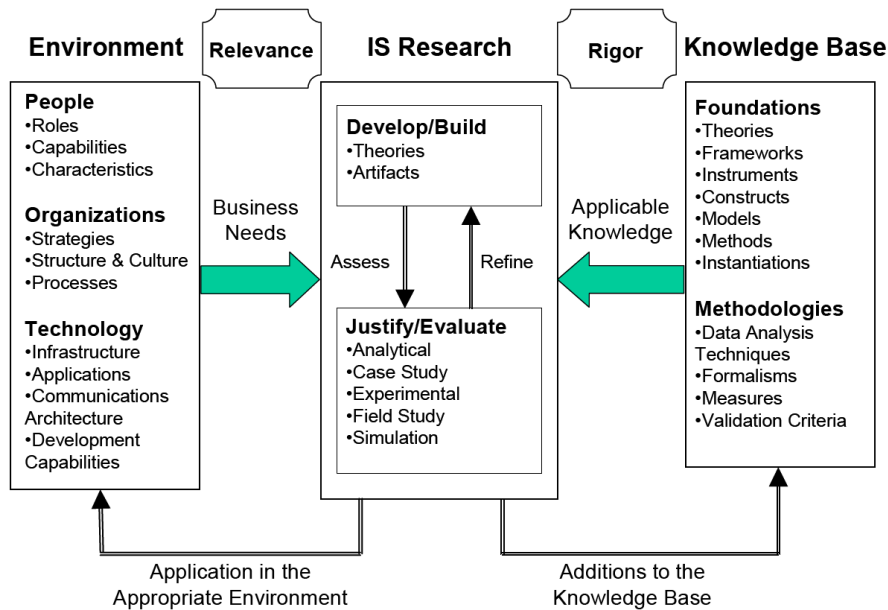


Figure 1.3.: Information systems research framework according to [He04].

continuously justified and adapted our solution based on the feedback we received from our industry partners. We also conducted a controlled experiment in order to examine the quality of our approach.

Since research “rigor is achieved by appropriately applying existing foundations and methodologies” this thesis follows scientific methods and standards as well as state-of-the-art development methods for constructing the artifact, as subsequently outlined.

Since “a justified theory that is not useful for the environment contributes as little to the IS literature as an artifact that solves a nonexistent problem”, the research presented in this thesis conforms with the guidelines for design science as introduced by [He04]:

1. **Design as an Artifact:** The research conducted in this theses results in an innovative extension of an existing Enterprise 2.0 platform. This extension facilitates the structuring of information and empowers users to flexibly adapt structured information. The introduced structuring mechanisms are also applicable to other Enterprise 2.0 platforms. Therefore, they can be considered as general means for applying structure to information in integrated Enterprise 2.0 environments.
2. **Problem Relevance:** Problem relevance is primarily derived from an analysis of the problem domain, that is, tool-based information structuring in enterprise contexts. The analysis considers approaches theoretically described in literature as well as those which are realized as working solution and used in practice. The literature analysis is introduced in Chapter 5, tool-based approaches have been evaluated in [BMN09, BMN10b, Da11, Mi10].
3. **Design Evaluation:** Utility, quality, and efficacy of the approach are demonstrated in Chapter 6. We show that the structuring concepts and mechanisms provided by the

extended Enterprise 2.0 platform are beneficial in practice. This is achieved by introducing several industry case studies. Additionally, the quality of the emerging structures is evaluated in a controlled laboratory experiment.

4. **Research Contributions:** The main contribution of this thesis consists of the developed artifacts. That is, a conceptual model supporting emergent information structures, a set of techniques facilitating information structuring, and an implementation based on an existing Enterprise 2.0 platform. Since these artifacts are applicable to similar contexts (e.g., the conceptual model can also be integrated in other Enterprise 2.0 platforms in order to support information structuring), the expanded, more general applicability can also be considered a research contribution.
5. **Research Rigor:** The evaluation of the research artifact follows scientific methods and standards. Utility is shown by means of case studies, quality via a controlled laboratory experiment according to [Kr95], and efficacy by providing a tool that allows to conduct quantitative analyses. The developed extension providing information structuring capabilities is based on Tricia, a generic framework¹ for model-driven development of domain-specific web applications supporting standardized web technologies [BMN10a, Bü07]. The process constructing the research artifacts follows an agile development approach [Ma03] in order to be able to directly collaborate with and receive feedback from users within the problem domain as well as to quickly respond to changes in the environment.
6. **Design as a Search Process:** We designed the artifact incrementally by applying several prototypes to different environments (e.g., in the Faculty of Informatics at Technische Universität München² or at our chair³). Furthermore, we established a community⁴ for enterprise architects [MN11b]. This community facilitates the exchange of experience in applying our artifact in the context of enterprise architecture management [La09]. We regularly gathered feedback from users from the problem domain. According to this feedback, we adapted our prototype and applied it to the problem domain again. We continuously applied this feedback loop during the research process. The thesis also draws conclusions from selected case studies (cf., Chapter 6) which encourage further research endeavors.
7. **Communication of Research:** In order to present the research results to a management-oriented audience we introduce selected industrial case studies. We describe which structures (i.e., domain models) emerge in practice (e.g., a domain model for the management of issues) by using the extended Enterprise 2.0 platform and how these structures are adapted. Furthermore, we show how individual users, teams, and enterprises benefit from our approach by demonstrating how the provided concepts and mechanisms help to fulfill business demands in structuring information (cf., efficiency in [He04]). The technology-oriented audience is addressed by introducing a model consisting of concepts that support information structuring (cf., language in [He04]) and by discussing how

¹The most prominent application build based on the web application framework Tricia is an Enterprise 2.0 platform. Therefore, the name Tricia is often used as a synonym for this platform. In this thesis, the name Tricia refers to the Enterprise 2.0 platform and not to the web application framework.

²<http://intranet.in.tum.de>; visited on March 10th 2012.

³<http://www.matthes.in.tum.de>; visited on March 10th 2012.

⁴<http://www.matthes.in.tum.de/wikis/sebis/wiki4eam>; visited on March 10th 2012.

these concepts are integrated in the Enterprise 2.0 platform Tricia with regard to the impact on the overall system architecture. Furthermore, we sketch the benefits of integrating our solution with the existing IT landscape of an enterprise.

1.4. Outline of the thesis

Chapter 2 presents the general ideas and design rationale underlying our approach. We sketch the notion of Hybrid Wikis by briefly introducing the core structuring concepts and by exemplifying how business users interact with these concepts.

In Chapter 3 we design concepts required to support information structuring in Enterprise 2.0 platforms. Based on these concepts, we show how information structuring is facilitated and how emergent information structures can be adapted by business users. To illustrate our approach, we present the main user interface views of Hybrid Wikis. This chapter addresses research questions 2 and 3.

In Chapter 4 we discuss how the concepts presented in Chapter 3 can be implemented based on an existing Enterprise 2.0 platform. Firstly, we explain the platform's underlying system architecture. Subsequently, we explain how Hybrid Wikis interact with this system from a technical point of view. This chapter answers research question 4.

Chapter 5 compares Hybrid Wikis to other tool-based approaches that support information structuring. We present the results of a literature study as well as findings from a survey on integrated Enterprise 2.0 applications. This chapter addresses research question 5.

In Chapter 6, which is concerned with research question 6, we validate Hybrid Wikis. We introduce case studies from different application domains and present our experience gained in six organizations. The quality of emergent structures is evaluated through a controlled laboratory experiment.

In Chapter 7 we respond to research question 1 by summarizing the key findings of this thesis. The thesis concludes with an outlook on future research directions.

In this chapter, we present the general ideas underlying our approach. First, we present the key considerations Hybrid Wikis are based on and briefly explain how users are intended to interact with the provided structuring concepts. Subsequently, in Section 2.1 the guiding design principles of Hybrid Wikis are introduced. In Section 2.2 we demonstrate the structuring capabilities of Hybrid Wikis by means of a small example scenario.

The main research objective of this thesis is to find a way to facilitate emergent and adaptive information structures in Enterprise 2.0 platforms. In particular, we want to encourage business users to provide structures and empower them to manage structures without the help of IT specialists. As currently Enterprise 2.0 platforms successfully support user-generated [O'05], user-managed contents, our goal is to extend these platforms to additionally support user-generated [HV09], content-driven, and user-managed structures. By empowering users to create and manage structures by themselves, the challenges discussed in Section 1.1 are mitigated.

The general idea is to make the contents in Enterprise 2.0 platforms accessible like in a database. The goal is to combine the characteristics of Enterprise 2.0 platforms (e.g., free-form content with hypertext and links; full text search) with those of database systems (e.g., structured tables with attributes and integrity constraints; structured queries) by integrating structured information in unstructured contents (cf., Figure 2.1). Our approach is called Hybrid Wikis. The term hybrid expresses that unstructured content is enriched with structured elements that allow to query content similar to a databases. Even if the name suggests that our approach is only applicable to wiki systems, it is possible to structure all types of unstructured content objects in Enterprise 2.0 platforms, such as blog posts, tweets, and documents. The name Hybrid Wikis originates from the first implementation of our approach, an extended wiki supporting content structuring integrated in an existing Enterprise 2.0 platform.

2. Towards Hybrid Wikis

Hybrid Wikis empower users to incrementally structure and to classify content objects and links using a small set of concepts. The core concepts are

- attributes,
- types, and
- constraints (cf., Figure 2.2).

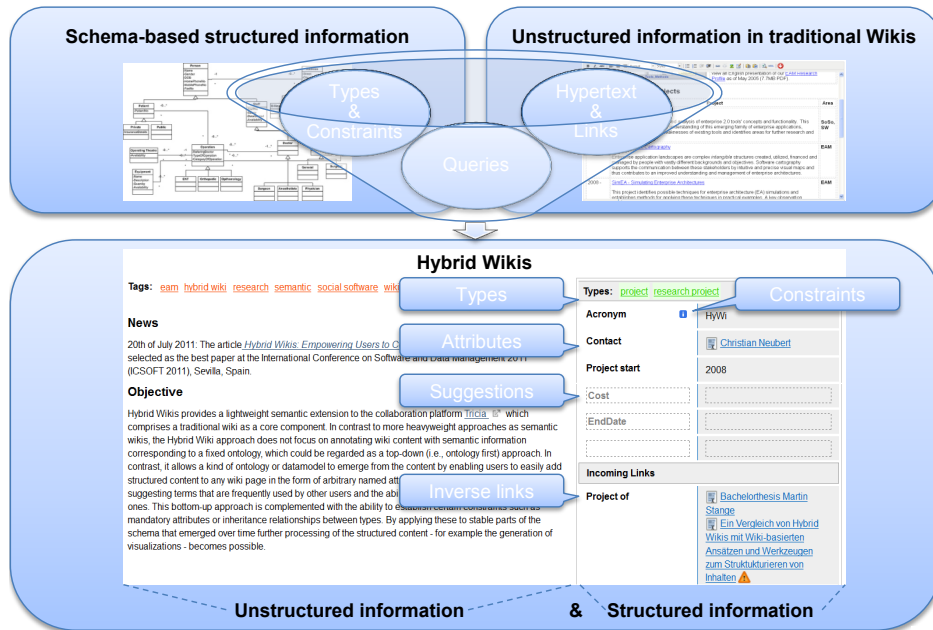


Figure 2.1.: Hybrid Wikis combine schema-based structured information with unstructured information provided by Enterprise 2.0 platforms.

Business users interact with these concepts using a familiar interface based on forms, spreadsheet-like tables, and autocompletion. Our assumption is that they feel familiar with these representations and in consequence are less inhibited in contributing structured information. Attributes are simple key-value-pairs that can be assigned to content objects, types provide means to classify objects, and constraints validate structured content. From these interactions, information structures (e.g., structured objects representing projects or customers) evolve as part of the users' day-to-day cooperative business activities, bottom-up as a byproduct (cf., Figure 2.2). For querying and browsing we provide a general search interface that allows a faceted drill-down based on the structured contents. By means of this interface, users can access emergent structures by using attribute- and type-based queries to generate custom views (e.g., lists, tables) and graphical visualizations (e.g., maps, graphs) of structured objects according to their specific information needs.

Business users can incrementally increase the structure's rigidity once a significant number of users provided structured content and a core information model has emerged. If required, they even can make these structures explicit and enforce them like an object-oriented database schema (cf., *Types & Constraints* in Figure 2.2). However, in contrast to a database schema in our approach users can change structures, even explicitly defined, at runtime without being

restrained by hard integrity constraints and data migration steps. The potential risk that structured information diverges from an explicitly defined schema is mitigated by generating suggestions (e.g., attribute suggestions based on schema information) and giving users the possibility to find and resolve inconsistencies. Furthermore, a set of techniques helps to unify and consolidate derivations in structures. The level of the explicitly defined structures' rigidity can be adjusted by the business users anytime (i.e., rigidity can be softened, if needed). This way, users are rather guided to follow an explicit information schema but never forced.

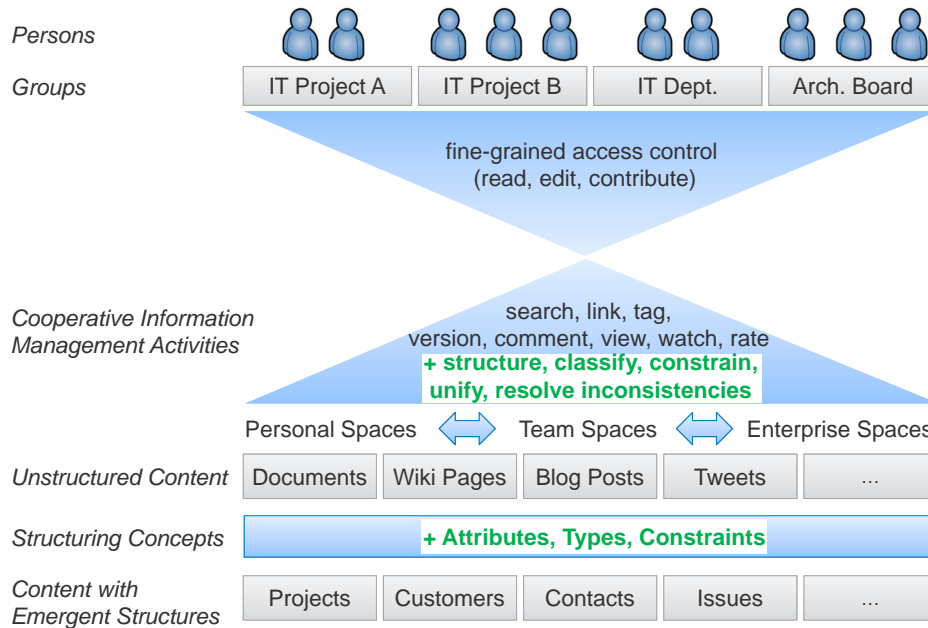


Figure 2.2.: Enterprise 2.0 platform extended with structuring concepts (attributes, types, constraints) provided by Hybrid Wikis.

2.1. Principles guiding the development of Hybrid Wikis

As described in Chapter 5, there are many other approaches (i.e., tools and prototypes) that already support various forms of information structuring in a collaborative web-based environment. We consciously decided to develop Hybrid Wikis not based on these tools or prototypes in order to explore a new approach to support information structuring in Enterprise 2.0 platforms by following a different set of guiding principles. In the subsequent sections, we make our design principles explicit. After the description of the design (cf., Chapter 3) and the implementation (cf., Chapter 4) of Hybrid Wikis, in Chapter 5 we compare Hybrid Wikis with other approaches to highlight the distinguishing, innovative aspects of our approach and some commonalities.

2.1.1. Simplicity over expressivity

This principle aims at lowering the barriers for non-expert users in contributing structured content. This means for using our approach neither special knowledge about the structuring concepts should be required nor should the user be forced to learn a query language to utilize the structured part of the contents. We try to find ways to enable all business users to enter structured data, in contrast to a two-phase process where unexperienced users enter textual content that is later enriched with formalized meta data (e.g., provided by the *Resource Description Framework* (RDF)) by experts. From our point of view, all attempts to translate between the expressivity of existing formalisms and the user by the means of new interfaces are unsatisfactory.

Hybrid Wikis primarily focuses on the user and accepts that there are limitations in the complexity of modeling concepts users can be expected to understand. We try to start from lightweight structuring concepts and metaphors that users are familiar with and then, in a second step, we examine how the data users provide by these simple means can be exploited by the system to offer features that usually require a formally defined data model.

Our approach mainly relies on simple keyword-like annotations of content, dynamically compiled and easily extensible forms for data entry, and the presentation of structured data in automatically generated tabular views. In turn, we try to avoid the notion of semantic annotations being something that is optionally appended to pure text content and that is defined in a separately maintained ontology or schema. Instead, we attempt to allow the user to implicitly provide semantics by filling data in particular fields of a form or a table, by optionally creating new such fields on demand, and by the way the data is queried and displayed in different contexts.

The principle *simplicity over expressivity* implies:

- A reduced set of structuring concepts instead of universal description languages.
- Default views to access structures instead of manual query-view configurations.
- Implicit management of information structures by means of views familiar to the users instead of formal and textual annotations.

2.1.2. Data over schema

Since we do not require the users to explicitly maintain a data model, we focus on dynamically mining the structured content to ‘guess’ the model and provide users with input options to guide them towards a consistent data model and vocabulary. The data model then emerges bottom-up from the cooperative information management activities of the business users. However, if required, advanced users can impose a schema and define certain integrity constraints.

Furthermore, we assume that it is even better that users provide invalid or inconsistent data than preventing them from data entry in order to satisfy restrictive schema definitions. Users should never be disturbed by hard integrity constraints during their daily cooperative information management activities. We rather provide means to detect inconsistencies by allowing

them to explicitly search for violations. Additionally, we increase the users' awareness by notifying them in case of violations via for example *Really Simple Syndication* (RSS) messages. Once inconsistencies are detected we provide mechanisms to harmonize derivations in structures by aligning data and schema.

The principle *data over schema* implies:

- Bottom-up, data-driven schema emergence instead of top-down meta modeling.
- Data first, schema second.
- Data gardening instead of data entry prevention.

2.1.3. Evolution over rigidity

Our approach focuses on structures that can flexibly be adapted. On the one hand, users are never prevented to change structured content according to their specific information needs, even in case of an explicitly defined information schema. We rather warn them if changed structures contradict any conventions. This way, structure continuously evolves, resulting in a 'vivid' information model. This model does not require any data migration steps when structure changes since it is directly derived (i.e., guessed) from the structured content objects. That is, the model always implicitly reflects the objects' structure. Additionally, users are never forced to enter information in a structured way, they are rather urged by providing a familiar user interface based on forms and spreadsheet-like tables.

On the other hand, if users impose a schema with integrity constraints, they are never forced to propagate changes in the schema to the data instances. Only if explicitly requested by the user schema adaption operations can be applied to instance data.

The principle *evolution over rigidity* implies:

- A vivid schema instead of rigid typing.
- Incremental structural adaption instead of large migration steps.
- Data migration on demand instead of data migration as obligation.

2.2. Example scenario

In the following, we illustrate the structuring capabilities of Hybrid Wikis taking the example of a small company's intranet. The example is originally described in [MNS11]¹, the article in which Hybrid Wikis have been introduced. We slightly adapted that article's example in this thesis since Hybrid Wikis have been developed further in the meantime.

Among other things, the company's intranet is used for gathering the knowledge about projects and people relevant for the company. We assume that while there may be many content objects (e.g., wiki pages) holding information about a person or project, there is one dedicated object for each such entity that can be considered the primary object which holds the basic

¹Some parts of [MNS11] are also used in Chapters 1 and 2.

information about it and optionally links to objects with more specific information. We further assume that in the beginning, there is only little content in the intranet and no types and attributes are used. As the number of projects increases over time, there is a growing demand for a more structured view on project related content. Attributes for project start and end dates are thus added to the respective objects and the type *project* is assigned. Having marked all project objects with the type, an overview table of all projects is instantly available showing sortable columns for the date attributes. Attribute values can be changed directly in this overview table. By this means, consolidating the project data is facilitated (e.g., adding missing information or standardizing the representation of attribute values).

Let's assume many of the company's employees have structured their profiles for themselves to provide some information about their specific skills and experiences. Some of them independently start to add attributes to the project objects to express their relationship with these projects, for example that they were members of the project team or project manager. In the beginning, people use different terms to describe their roles. As these inconsistencies become visible in the overview table, they are quickly harmonized by the users. It is not necessary to navigate to the respective objects since the table cells can be edited directly. As a result, if somebody now creates an object and assigns the type *project*, she is offered to provide values for the attributes *start*, *end*, *manager* and *member*. If she now adds a link to her profile to the *manager* attribute, this reference will be automatically visible in her profile in the *incoming links* section. A new entry *manager of* with a single value being the link to the project is displayed (analogous to *Project of* depicted in Figure 2.1). If she is already manager of other projects, the list will simply be extended by one entry. Additionally, she decides to reference the other project members in order to have their contact information near by, records the bonus persons involved additionally received for their work, and documents the date the members started working in the project.

Finally, she relates the project to the ordering customer and to the primary customer contact for this project. The primary contact is additionally flagged to express that this person is a decision maker. The modeling of customers and external contacts can be considered as a simple *Customer-Relationship-Management* (CRM) system. This brings benefits since those pages can even be referenced in other contexts, resulting in a web of knowledge. For instance, meeting notes can include links to the participants by referencing pages describing customer staff.

Starting from this basic schema, the company can further adapt it to new needs: Besides adding more attributes, the existing ones can be refined. The attribute *manager* can be made mandatory for the type *project* so users are additionally reminded to provide this attribute. If it is omitted, the object is flagged as invalid. It can be further specified that only a single link to an object having the type *internal* is accepted as a value. New types can be added to distinguish different types of projects like *internal project* or *research project*. On the one hand, this allows the generation of lists and tables containing only the respective subset of the projects, on the other hand, the system is supported in offering the user more relevant attribute suggestions (cf., Figure 2.1) when editing structured data. For example, for a research project the attribute *area* could be suggested whereas attributes only relevant for internal projects are not shown. It is also possible to add a temporal dimension to the project types by adding types like *project in preparation*, *current project*, or *completed project*. Using types instead of a status attribute has the advantage that attribute constraints can be related to the types.

For example, each object of a completed project can be required to contain the actual end date of the project. However, it is also possible to define an enumeration constraint that only allows specific values to be used, such as *planned* and *active*.

The model underlying this example is depicted in Figure 2.3. The example is used to illustrate Hybrid Wikis in the subsequent chapters of this thesis.

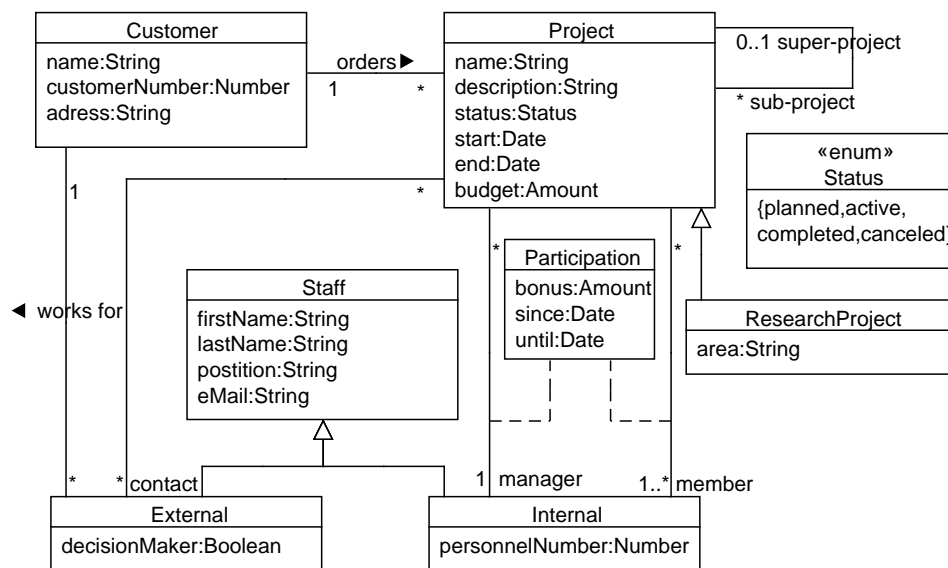


Figure 2.3.: Exemplary information model used within a small company's intranet for project and customer management.

In this chapter, we present our approach, Hybrid Wikis. In Section 3.1, we present the core concepts required to support information structuring in Enterprise 2.0 platforms. We incrementally introduce the conceptual model underlying Hybrid Wikis and discuss important design decisions. Based on this model, in Section 3.2 we discuss how information structuring can be integrated in Enterprise 2.0 platforms, in Section 3.3 we describe how users interact with these concepts. In particular, we show how information structuring is facilitated and how structures can flexibly be adapted. To illustrate our approach, in Section 3.4 we present screenshots of the current implementation of Hybrid Wikis.

3.1. Concepts supporting emergent information structures

In this section, we introduce the concepts underlying Hybrid Wikis. The core model is depicted in Figure 3.1. This model can be considered as the conceptual design of Hybrid Wikis and serves as the basis for the implementation as introduced in Chapter 4. We explain the model's concepts step-by-step, discuss important design decisions, and sketch some alternative solutions. The core model shown in Figure 3.1 is extended incrementally by introducing additional concepts necessary to support emergent information structures. Concepts supporting the structuring of concrete content objects are colored in orange, concepts representing information structures on a meta level are colored in green. Grey colored concepts represent elements built-in given by the underlying Enterprise 2.0 platform, that is, elements not developed as part of this thesis. We consider the model as orthogonal applicable to Enterprise 2.0 platforms. However, a concrete implementation most probably requires to slightly change it according to the platform's underlying architecture. In Chapter 4, we explain which changes we applied in order to implement the model based on the Tricia¹[BMN10a] architecture.

¹<http://www.infoasset.de>; visited on January 2st 2012.

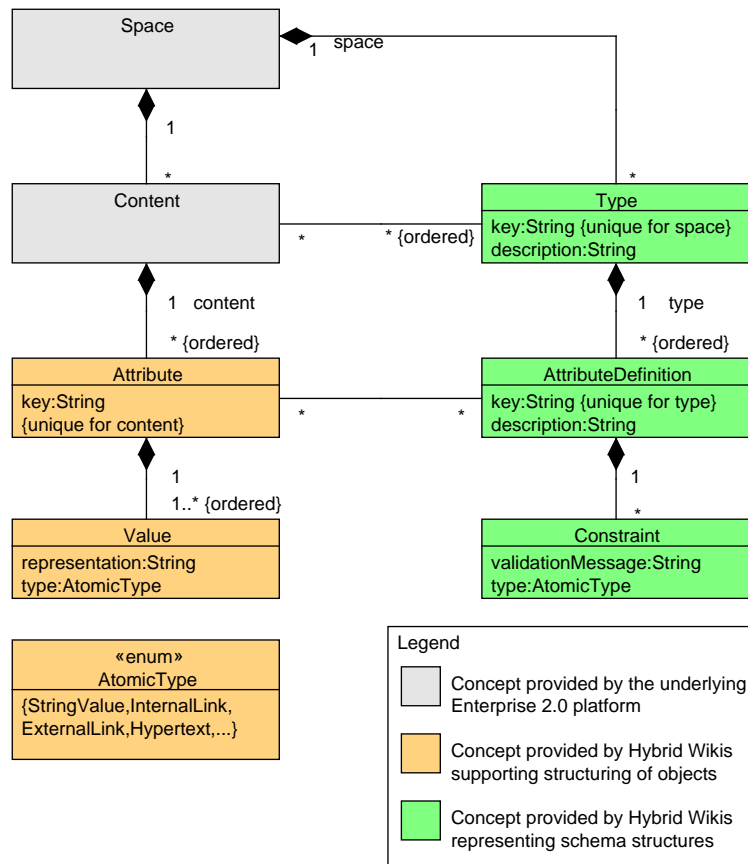


Figure 3.1.: Core concepts provided by Hybrid Wikis.

3.1.1. Spaces

A space primarily serves as a container for a set of content objects (cf., Section 3.1.2). It typically encapsulates those kinds of objects which are homogeneous with regard to their context, for example objects which are modified within a specific group of person for purpose of collaboration. Therefore, a space can be considered as the namespace [Te09] for its objects. This means each content object can only be unambiguously identified within a space's context. Additionally, a space can be used to define defaults for its containing objects, such as default access rights (e.g., default readers cf., Figure 3.4) for newly created objects (cf., Section 3.1.2) or to generate overview lists of the contained objects shown in the *User Interface* (UI). Furthermore, it provides a unique name as well as a *Uniform Resource Locator* (URL). The URL typically is an encoded representation of the space's name by default (cf., URLs in Section 3.1.2). The space with its properties is depicted in Figure 3.4.

3.1.2. Content objects

Content objects refer to objects containing unstructured user-generated information [O'05]. The term and its definition are primarily taken from [BMN09], in which a unifying multi-

dimensional classification and evaluation framework of Enterprise 2.0 platforms is presented. For instance, content consists of plain textual information or embedded rich media objects, such as videos. A content object is located in exactly one space, which represents the container for a specific set of objects in the application (cf., Section 3.1.1). Typical content objects are for example wiki pages and blog posts.

In many Enterprise 2.0 applications content is represented by and stored in form of a markup language. Typical markup languages are wiki markup [AMY08]² or simple hypertext [BLC95]. In order to prevent users directly getting in touch with that markup editing is supported by a so-called WYSIWYG Editor [Du05]. Such an editor allows to manipulate the markup by means of a graphical user interface and serves as a more user-friendly abstraction. The content object with its properties is depicted in Figure 3.2.

User-generated in this context means that the content is provided and maintained by the individual users themselves and not statically given by a third-party web provider. Furthermore, user-generated content here differs from that in content management systems [Bo04] since it is immediately visible on the web site instead of being reviewed and revised from other users for reasons of quality assurance in a separate process before publication.

Moreover, we assume that content is primarily created in two kinds of contexts: in a collaborative environment in which different users closely work together on shared topics, such as in wiki applications for collaboratively work on shared articles, and in an author-centric environment in which authors usually produce contents without the intention of collaborating with others, such as in blogging platforms.

Besides the information's plain unstructured parts (e.g., text and embedded media objects), content can consist of semi-structured elements, such as links or tables (i.e., hyperlinks and HTML tables in case of using *Hypertext Markup Language* (HTML) as markup language). Semi-structured in our model means that there is no additional information explicitly given about the meaning of the data, such as semantic annotations [KVV06]. This means plain textual unstructured content including semi-structured elements (e.g., links and tables) and structured elements (cf., Section 3.1.3) are clearly separated from each other. That is, markup only consist of pure information (and in case of HTML as markup language some information regarding the text formatting, e.g., bold) and is not additionally enriched with meta data representing the structured parts, such as in [KVV06]. In this thesis, the content object's markup containing unstructured as well as semi-structured information is referred to as the built-in markup or full text respectively.

3.1.2.1. Links

We distinguish two kinds of links within the built-in markup. Internal links referencing content objects within the application and external links pointing to resources outside of it. Furthermore, we distinguish outgoing and incoming links. Outgoing links are internal or external links pointing from the built-in markup to other resources, whereas for incoming links this is exactly the opposite way around, that is, internal or external links that point to a content object from the full text of other objects. Incoming links in this thesis are used as synonyms

²<http://wikicreole.org>; visited on February 1st 2012.

for backlinks or inverse links respectively. Furthermore, for reason of simplification we use the terms link and internal link synonymously.

A link defined in a specific context we refer to as a contextualized link. For instance, a context can be a semi-structured element, such as a hypertext table embedded in the built-in markup, or a link value of a structured attribute (cf., Section 3.1.5). In many Enterprise 2.0 applications this context is shown as part of an incoming link. For example, if a link is embedded in a hypertext table providing a header row the incoming link of the referenced object could show the column name from that header as the context in addition to the link to the referencing object. If no additional context is specified, we refer to it as an anonymous link, that is, the context of an anonymous link is the content object itself. Except external incoming links, all kinds of these links can be maintained by the application itself and are treated as first class objects.

In Enterprise 2.0 applications internal incoming links are often explicitly shown for an individual content object. For instance, the degree of incoming links can serve as an indicator for the importance of a resource. Furthermore, readers of a content object get an overview in which other contexts it is used and additionally the navigation is facilitated. Internal links between content objects (i.e., from the markup of a content object to another internal content object) are depicted in Figure 3.2 indicated by the role named *links*, internal incoming links are represented by the role *referenced by*, and internal outgoing links by the role *references*. External outgoing links are not depicted there. However, in the simplest form they would be represented as a set of URLs in our model.

3.1.2.2. Access control

In contrast to traditional Web 2.0 applications [O'05], such as wikis, in an enterprise environment it is particularly important to protect certain information from unwanted access (e.g., for reasons of confidentiality). Therefore, for all content objects discretionary access control mechanisms can be applied. This means for an individual content object can be specified who is allowed to see it (i.e., read access) and who is allowed to apply changes to it (i.e., write access) by using an *Access Control List* (ACL) [BMN09, PT11]. Some applications provide additional access right roles, such as contributors which are allowed to create new content objects within a space. Figure 3.2 depict the access rights model underlying this thesis in order to protect objects from unauthorized access. The model corresponds to that used in the web collaboration system Tricia. However, we could also apply a different access control model. In Section 3.2.3, we discuss how access control mechanisms impact information structures.

The *readers* role specifies the principals which are allowed to see the content object and the *writers* role the principals which are authorized to modify it. In this access control model, a principle serves as an abstraction of persons and groups. Figure 3.3 shows the complete access control model consisting of persons, groups, and principals. A membership as depicted in this figure contains additional information about the current state of the relation between a group and its person (e.g., the date a person has joined a group). It does not have an own identity since its existence depends on the life-cycle of the associated group and person. Both kinds of principals (i.e., readers and writers) are globally available in the application. The privilege to create new principals can be limited to specific groups within the organization.

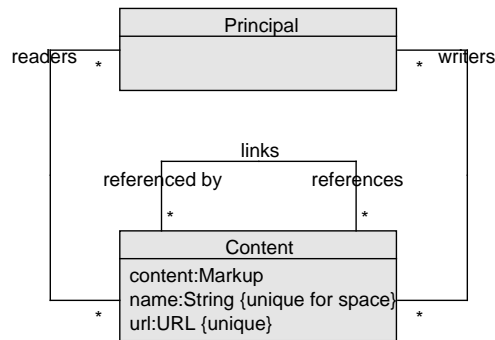


Figure 3.2.: Access control for content objects.

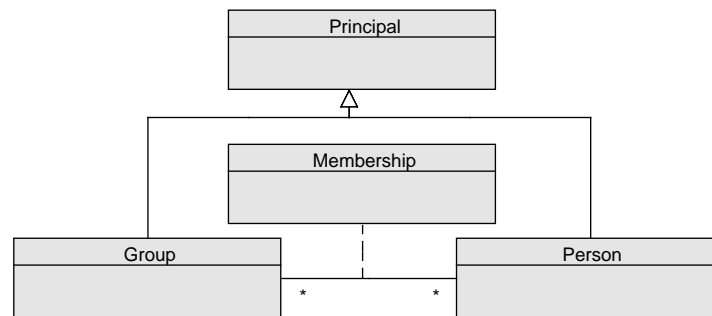


Figure 3.3.: Access control model according to [BMN10a].

In some Enterprise 2.0 applications the access rights of a space also serve as a default for the contained content objects. Otherwise, it would be tedious to maintain access rights for each content object individually. This means if a new content object is created, the access rights from the corresponding space are passed to the new object (cf., Figure 3.4). However, in these applications individual access control lists (with readers and writers) can be defined for objects by overriding the space's default settings. Readers and writers in this model are disjoint, that is, a principal (person or group) can either be a reader or a writer. However, it is obvious that writers are also allowed to read content objects. This means that the principals from the readers role are always additionally specified with regard to the writers role.

3.1.2.3. Identity

Each content object provides a textual name, which in most cases corresponds to the page title or to the caption being shown on the web page respectively. For instance, in the Wikipedia encyclopedia³ the name corresponds to the article's topic which also is used as the page's caption. In the Wikipedia project the name is unique within the entire application, in the model as depicted in Figure 3.1 the name of a content object is unique with regard to other content

³<http://www.wikipedia.org>; visited on January 2st 2012.

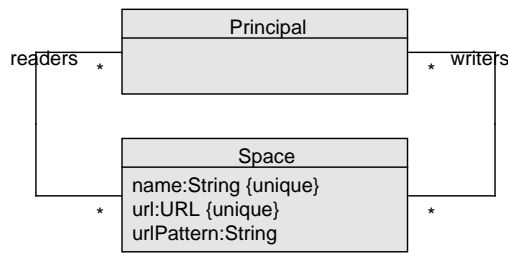


Figure 3.4.: Default access rights derived from a space.

objects within its associated space. This means that the name also serves as a conceptual identity within the space it belongs to. The content object's name is depicted in Figure 3.2.

In order to provide uniform access to a content object it provides a human-readable, stable, and unique URL. Since content in Enterprise 2.0 applications is frequently modified by the users, such as wiki pages in wikis, these applications typically provide version control support. By doing so, it is possible to trace the evolution of content and to revert changes, if necessary. Therefore, stable in our context means that each individual version of a content object is accessible by a URL, which will never change. This means a resource underlying a stable URL can be referenced even outside of the application without being concerned about losing the context caused by changing content. For instance, in this thesis the date of access is always given when referencing a web resource by URL since the underlying website might have changed when reading this thesis. This would not be necessary if stable URLs would be provided. Another example is the usage of URLs as bookmarks in the web browser. However, a URL is only stable within the life-cycle of the content object, that is, even a stable URL becomes invalid if the associated content object is deleted. Stable URLs in literature are referred to as permalinks [Li09], which is shortened from two words: permanent links.

A permalink referencing a specific version of a content object is only partially human-readable since it needs to contain information about the individual version, which in most cases consists of a numeric identifier. Therefore, the URL of the latest version of the content object often is shortened to a human-readable representation which does not contain any information about the version history. For instance, the URL of the Wikipedia article explaining permalinks is represented as <http://en.wikipedia.org/wiki/Permalink> and always points to the most latest version of this article, a permalink of a specific version is represented as <http://en.wikipedia.org/w/index.php?title=Permalink&oldid=453810449>, whereas the *oldid* attribute contains the specific version information. Considering applications without version control capabilities a stable URL implies that the content of the underlying resource never changes over time.

Since some applications allow to modify URLs of a resource, stable in our model also means that the application needs to provide mechanisms to ensure that all other objects pointing to this resource do not become outdated afterwards, that is, the URLs of these referencing objects stay valid after renaming the referenced resource. These mechanisms have to consider both kinds of referencing objects, internal content objects, which are maintained by the application itself, as well as external objects, which are persisted in different applications, for example

bookmarks stored in the web browser of a user. Internal objects can be directly updated by the application, for external objects different mechanisms need to be applied, such as moving the original content to the object with the new URL and leaving a unmodifiable stub having the old URL saying that the original content is accessible at a different URL now.

Human-readable here means that the URL mainly consists of meaningful words giving users of the application an indication of the resource's content. Typically in Enterprise 2.0 applications the URL corresponds to an encoded representation of the content object's name by default. For instance, the URL of the Wikipedia article describing a uniform resource locator is encoded as `http://de.wikipedia.org/wiki/Uniform_Resource_Locator`. If the path⁴ of the content object's URL is composed of the encoded name of the object itself and the encoded name of its containing space (cf., Section 3.1.2), the path is unique with regard to its associated space by default (i.e., for newly created content objects), just the same as for the uniqueness of the content object's name. This means that the URL path can also serve as a conceptual identity for a content object within the space it belongs to, even if the URL might change within the content object's life cycle. For example, the default encoded URL path of a wiki page with name *a b* in space *x y* would be represented as `/wiki/x_y/a_b`. The content object's URL is depicted in Figure 3.2.

Content objects provide an arbitrary ordered set of attributes (cf., Section 3.1.3). By means of attributes objects can be structured.

3.1.3. Attributes

In their simplest form, attributes are key-value pairs that can be added to content objects. They consist of an attribute name – the key – and a value (cf., Section 3.1.5). The key is a simple character sequence which is depicted in Figure 3.1. Furthermore, an attribute is unique with regard to its key and within its corresponding content object. The attribute key can serve as the default ordering criterion for the attributes of a content object, for example by default attributes are sorted alphabetically by key. However, the order of attributes can diverge from the default order.

An attribute belongs to exactly one content object. Attributes in our model do not represent meta data but constitute the structured part of the content object. For instance, a wiki page describing (i.e., in the full text of the page) the purpose of a customer project *Development Project A* within an enterprise additionally provides an attribute *Budget* indicating the amount of money available. In our model this attribute is not part of the content's markup integrated as annotated meta data, it represents the content's structure separately.

Individual attributes provide an ordered set of values (cf., Section 3.1.5). This set contains at least one element, that is, an attribute cannot exist without a value.

3.1.4. Types

In our model a type serves as a classifier for content objects. For instance, the type of the object describing *Development Project A* would be *project*. Additionally, a type has a

⁴<http://www.w3.org/Addressing/URL/url-spec.txt>; visited on November 1st 2011.

description in order to make a statement about its meaning. For example, the description can be used to give users a hint regarding the type's meaning before assign it to a content object. The type with its properties is depicted in Figure 3.1. The key represents the type as a simple character sequence (i.e., in our example *project*). Types can be shared across multiple content objects, but not among different spaces. An individual content object can be related to an ordered set of types, called type assignments in the following. Content objects with types we refer to as typed content objects. Typed content objects represent instances of that type. In some Enterprise 2.0 applications the types are sorted alphabetically by default. However, the order can diverge from the default for individual content objects. Types in our model do not represent meta data but constitute the structured part of the content, similar to attributes.

It is important to note that types are basically independent of attributes. Independent here means that the content object's types can always be changed without having an impact on the object's attributes. For instance, a type can be removed from a content object without removing its attributes and attribute values (cf., Section 3.1.5). Conversely, this also means that attributes can arbitrarily be added to or removed from a content object. Furthermore, types can also exist without the existence of a content object having this type assigned. In this way, types and attributes can be flexibly combined with each other and therefore also be considered as a more flexible alternative to template-based approaches, such as introduced in [HLS05].

This flexibility implies that in our model content objects having the same set of attributes not necessarily belong to the same type. While this is not quite unusual, this also means that different content objects can have the same type but differ with regard to their attributes, that is, in the most extreme case two content objects with the same type can have a completely disjoint set of attributes. If these types are intended to have same semantics, this fact contradicts the idea of types - encapsulating objects with similar properties. However, in Sections 3.3 and 3.1.7 we introduce several techniques in order to counteract this disparity of types and attributes.

Furthermore, it is important to mention that on the one hand it is consciously avoided to explicitly create relationships between types and attributes as it would be done when defining a template [HLS05] or in classical type-oriented approaches such as semantic wikis [Vö06]. On the other hand, in some cases it is still reasonable to assign a list of attributes to a type, for example in order to specify integrity constraints for an attribute in the context of a type, like the number of values or allowed value ranges. In Sections 3.1.7 and 3.1.8 we explain how attributes can be bound to a type in order to specify integrity constraints without losing the flexibility of our approach.

3.1.5. Values

A value is the second part of a key-value pair as introduced in Section 3.1.3, it belongs to exactly one attribute. Furthermore, all values provide a textual representation (i.e., can be represented as a simple character sequence). Regarding the project example from Section 3.1.3 *Budget* would be represented by the value *200.000*. That character representation can serve as the default ordering criterion for the values of an attribute (e.g., by default the values

are sorted alphabetically). However, the values' order can diverge from the default order for individual attributes.

Additionally, each value has a specific data type, such as string or link. The enumeration *AtomicType* (cf., Figure 3.5) illustrates possible data type variations. For instance, the type of *Budget* would most likely be integer. Since the data type is directly bound to an individual value and not to the corresponding attribute the values of an attribute can differ regarding their types. For instance, an attribute can have two values, one with data type string and the other with data type (internal) link.

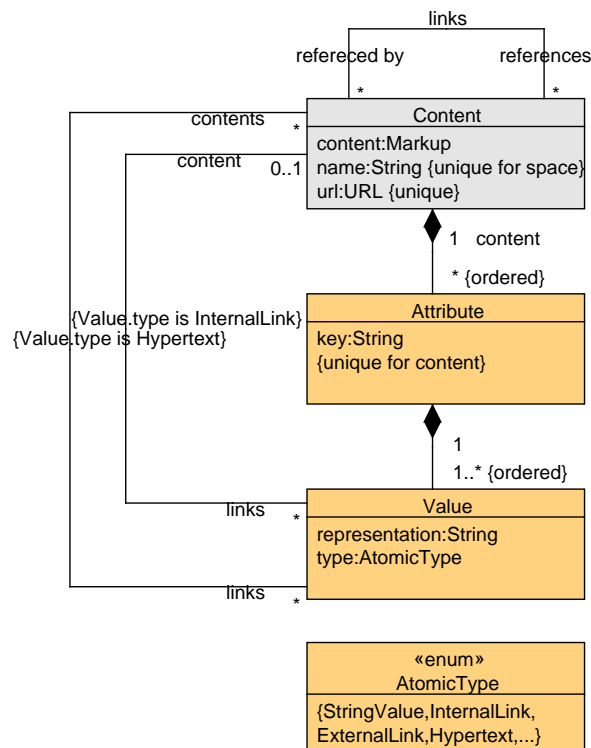


Figure 3.5.: Concepts supporting the structuring of content objects.

In the following, we sketch the most important data types in the context of our model. The value's type attribute and the elements of the enumeration *AtomicType* are replaced by corresponding subclasses, one class for each individual data type. Subsequently, the specific characteristics of each data type are highlighted. We replace the value's textual representation by a more data type-specific one for purpose of illustration. However, it is still necessary to have different representations of an individual value (especially that a value can be represented as a simple string). For instance, the representation in the user interface differs from the internal representation in the storage. This is discussed in Chapter 4 in detail.

3.1.5.1. StringValue

In the simplest form a value is a string literal, that is, a simple character sequence (cf., Figure 3.6). String is a value's default data type since this is the most common one. Besides

strings, many further literals can be imagined. Especially in the context of enterprises we consider the literals number and date to be important. Since attributes can consist of multiple values with possibly different data types, it would be difficult to sort these values correctly (e.g., in a table view showing multiple attributes with the same key) if we would not distinguish between strings, numbers, and dates. Date and number are not explicitly depicted in our model since they provide similar characteristics as a string value. We are aware that the definition of number and date as given here is not very precise. For instance, a number could either be an integer or a floating-point number. In the latter case it would be necessary to specify the number's precision. However, in Chapter 4 we explain how the different characteristics and formats of values can be handled.

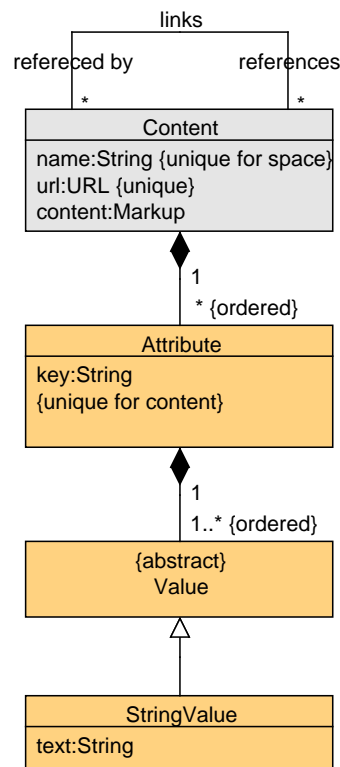


Figure 3.6.: String values.

3.1.5.2. InternalLink

An internal link is a link value referencing a content object. The difference to an internal link as introduced in Section 3.1.2 is that the context of an internal link is always given by its corresponding attribute. For instance, the name of this attribute can be shown in addition to the link to the referencing object in order to give the users more precise information about the context of the incoming link. Incoming links originating from an internal link value we refer to as structured incoming links in the following. Since the referenced object can be deleted independent of the link value an internal link is either pointing to exactly one content object

or to none (cf., Figure 3.7: the cardinality is defined as (0..1)). The internal link is depicted in Figure 3.7.

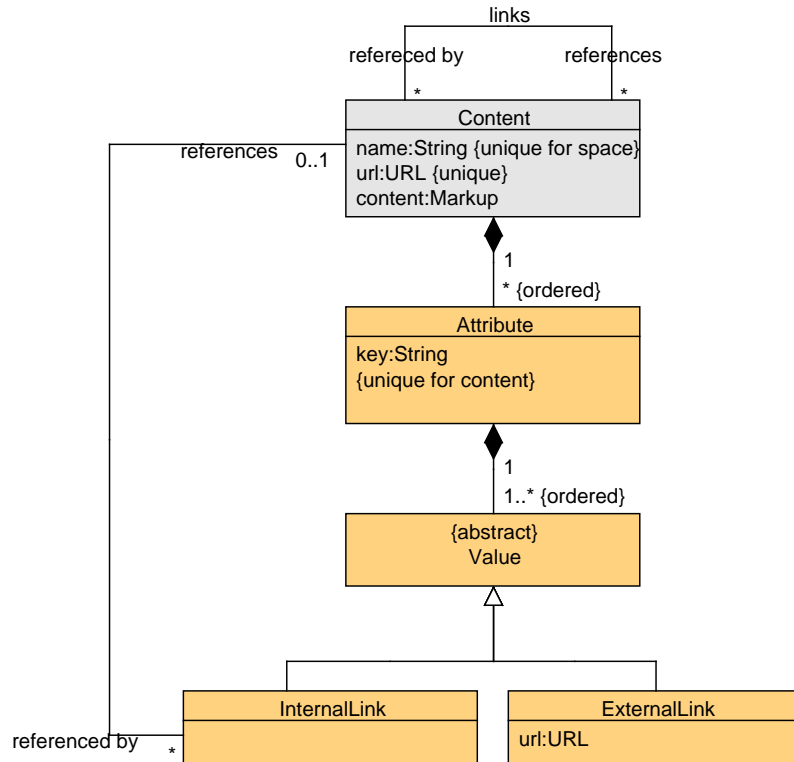


Figure 3.7.: Internal and external link values.

3.1.5.3. ExternalLink

An external link is a link value referencing a resources outside of the application. The difference to an external link as introduced in Section 3.1.2 is that the context of an external link is always given by its corresponding attribute. The external link is depicted in Figure 3.7.

3.1.5.4. Hypertext

A hypertext value contains arbitrary unstructured as well as semi-structured information, but does not contain further structured elements, similar to the built-in markup as introduced in Section 3.1.2. The difference to the built-in markup is that the context of a link within the markup is always defined by the markup's attribute. However, the context of these links can be more specific, for example if a link is embedded in a (HTML) table. A markup value can also contains external links, but are not depicted in the model. The hypertext is depicted in Figure 3.8.

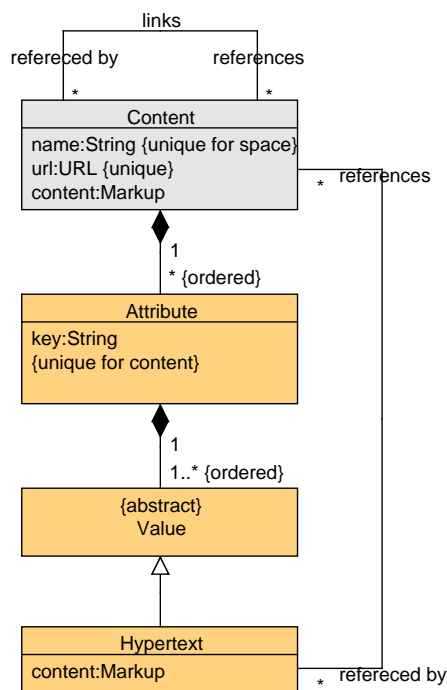


Figure 3.8.: Hypertext values.

3.1.5.5. Record

A record value is an ordered list of key-value pairs consisting of at least one pair. Record values can also be nested, that is, the attribute list of a record value can contain an attribute which itself contains record values. Since it might cause difficulties when evaluating record values, it is not allowed that the record's owning attribute is contained in the list of the record's attributes. For this reason the relation between record and attribute is declared as *acyclic*. This restriction also applies to transitive relations between attributes and records (i.e., for nested records). For instance, if an attribute a_1 having a record value which contains an attribute a_2 also having a record value then a_2 's record must not contain attribute a_1 . The record's attribute list is existentially dependent on the record, that is, all attributes are deleted if the record is deleted. Additionally, a record value can be typed, that is, it can have an ordered set of types (cf., Section 3.1.4).

Since a record consists of structured attributes and types it is very similar to the structured part of the content object. The difference is that a record does not provide a built-in markup property and only exists in the context of its owning content object, that is, it does not have an own identity and cannot be referenced by a URL. Therefore, a record value can be considered as an anonymous object within a content object. The record is depicted in Figure 3.9.

Regarding the *Development Project A* example from Section 3.1.3, additionally an attribute *Customer Address* might be given. This attribute could be represented as a record consisting of the two attributes *Street* and *Street number* typed with *Address*.

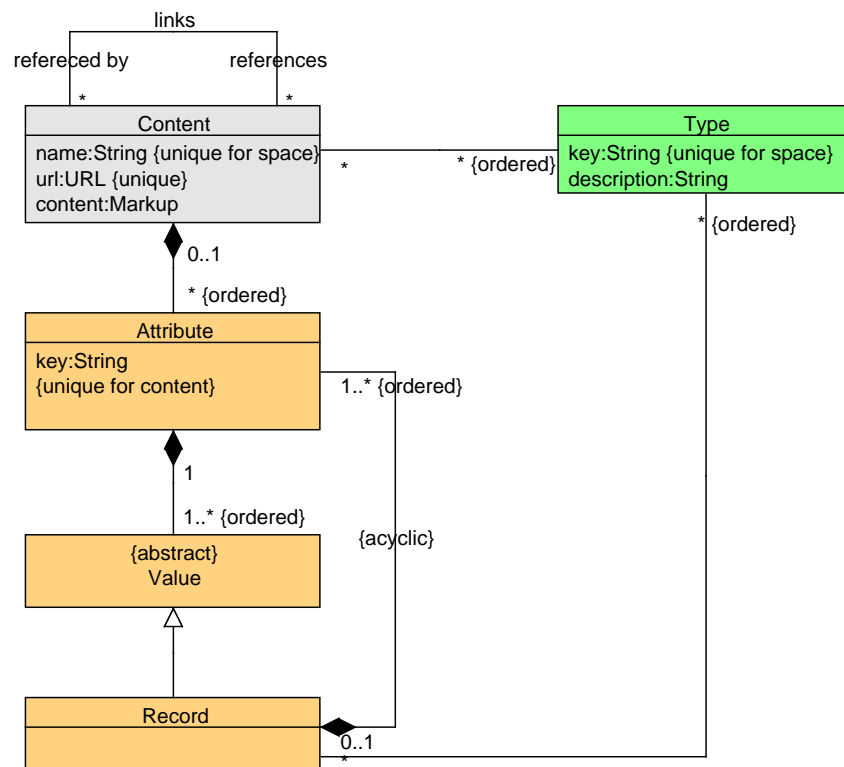


Figure 3.9.: Record values.

3.1.5.6. StructuredLink

A structured link is an internal link, that is, a link referencing a content object. Additionally, a structured link consists of structured attributes and types, similar to record values. Thus, a structured link can also be considered as an anonymous object, but in this case its existence primarily depends on the relation between two content objects. This means, if the link between these objects is deleted, the types and structured attributes are deleted as well. The structured link is depicted in Figure 3.10.

Regarding the *Development Project A* example from Section 3.1.3, additionally an attribute *project lead* might be given referencing a content object *Max Leader*, which is a person working in the company. This attribute could be represented as a structured link additionally providing an attribute *bonus* in order to express the amount of money he additionally receives for leading the project. In this example, the information about his bonus gets lost if the project lead changes since the link to *Max Leader* is replaced by another one. In Section 3.3 we introduce a technique in order to prevent that this additional information gets lost if a structured link is replaced or deleted.

Besides internal links, external links can also provide structured attributes and types. But since external structured links have the same characteristics as internal structured links they are not further detailed in our model.

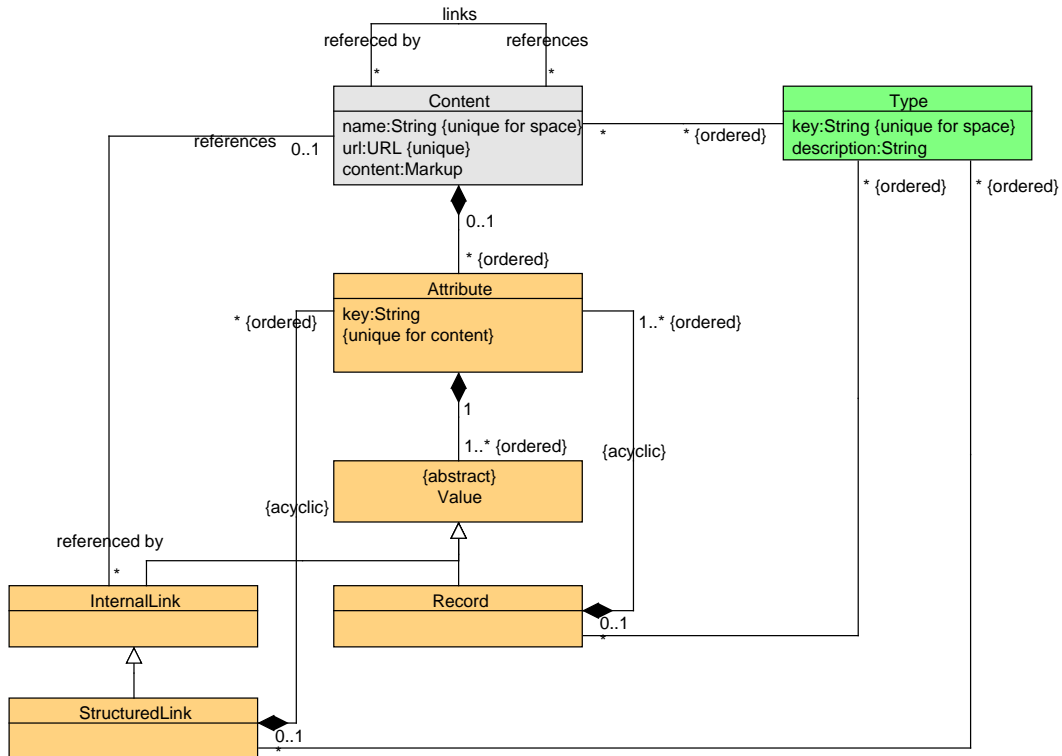


Figure 3.10.: Structured link values.

3.1.6. Structured

As mentioned in Section 3.1.5.6, the data types structured link and record are quite similar since both provide a list of attributes and an ordered set of types. The same applies to the content object, it also provides both relations, to types and to attributes (cf., Figure 3.10). In order to simplify the model’s representation we subsume these common properties (types and attributes) of content object, structured link, and record to one concept, namely structured (cf., Figure 3.11). This concept represents the ability of an object to capture structured information (i.e., attributes and types). We return to this simplified representation of the model in Chapter 4, but do not refer to it in the following. Since in this simplified model a structured object consists of an ordered set of attributes of any size, we have to ensure that at least one attribute is specified when an object of type structured belongs to a record value (since a record provides at least one attribute). This could be ensured by for example providing an *Object Constraint Language* (OCL) expression, but is not depicted in Figure 3.11 for reasons of simplicity.

3.1.7. Attribute definitions

As discussed in Section 3.1.4, the flexibility of assigning types and attributes independently can lead to a mismatch between them. Attribute definitions facing this problem by allowing a fix assignment of attributes to types. Fix in our model does not mean that the connection

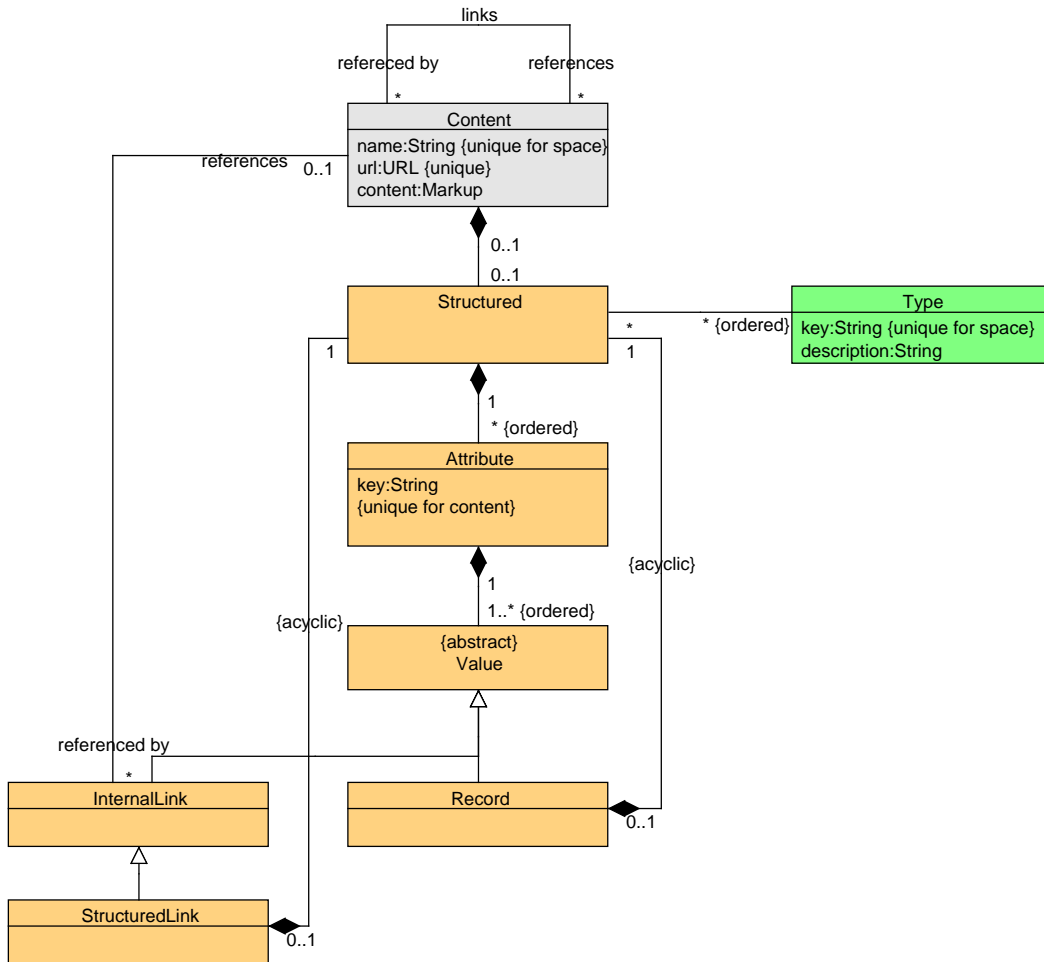


Figure 3.11.: Simplified model representing structured content objects.

between them is immutable, like in a schema of a relational database [Ha09], but rather that the attributes are loosely associated with a type and can be changed anytime. Furthermore, the declaration of attributes for individual content objects is still optional even if the content object is related to a type with a fixed set of attribute definitions.

An attribute definition belongs to exactly one type and has a key which is unique with regard to its corresponding type. Additionally, an attribute definition has a description in order to make a statement about its meaning. For instance, the description can be used to give users a hint regarding the attribute's meaning while assign it to a content object. The attribute definition with its properties is depicted in Figure 3.12. An attribute definition conforms to an attribute of an individual content object by type and key. This means, an attribute of a content object is related to an attribute definition if both have the same keys and if the content object is typed with the same type the attribute definition belongs to. Since multiple types can be assigned to a content object an attribute can also be related to multiple attribute definitions. However, attributes and attribute definitions can only be related if the content object is typed.

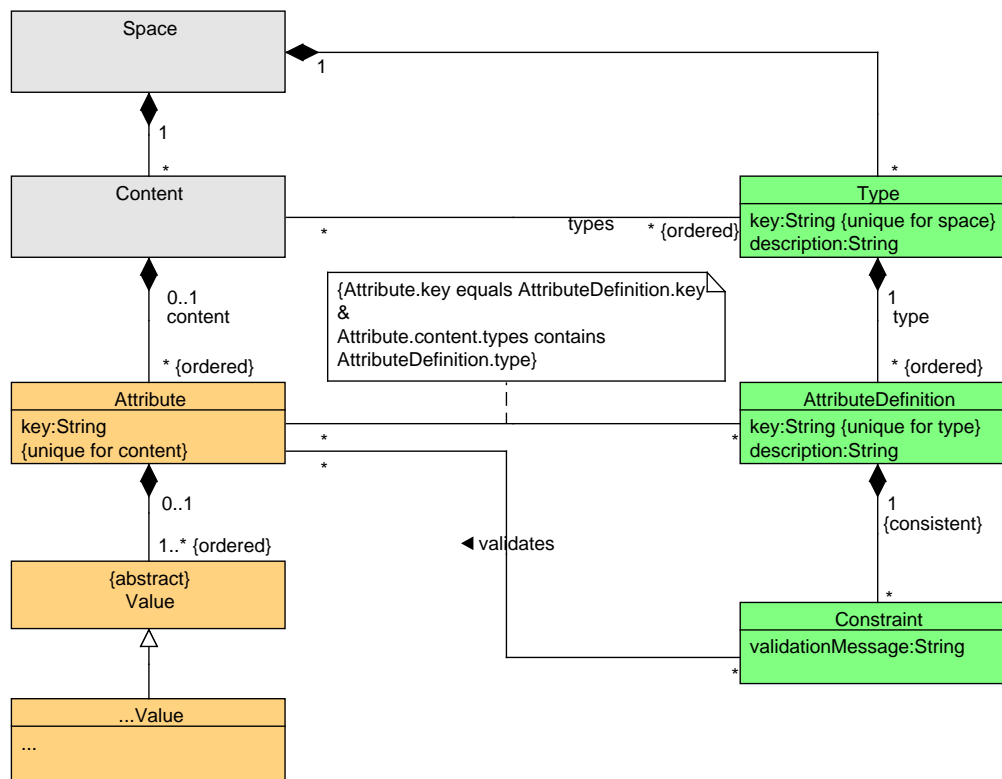


Figure 3.12.: Core concepts specifying types and integrity constraints.

Although attribute definitions allow a stronger binding between attributes and types, they alone are not enough to ensure a more consistent usage of attribute-type combinations for individual content objects. For that reason we introduce the technique of attribute suggestions later. For example, attributes are suggested for individual typed content objects by means of corresponding attribute definitions. Attribute suggestions are introduced in Section 3.3.1.

As indicated in Figure 3.12, the attribute definitions of a type are ordered. In most cases the single-typed content objects should have the same attribute order as given by the order of the type's attribute definitions. However, the attribute order of individual content objects can diverge from the order given by the related definitions (cf., Figure 3.12). In Section 3.3 we explain how different attribute orders can be consolidated, in Chapter 4 we introduce how they are implemented.

Besides the closer binding of types and attributes, attribute definitions are used to validate the content object's attributes and values by means of integrity constraints (cf., Section 3.1.8). Any number of constraints can be assigned to an attribute definition. A constraint applies to an attribute and its values if the attribute is related to its attribute definition, that is, both are using the same key and the same type. If the attribute or its values do not conform to the rules of the constraint, the constraint's validation message is used to indicate this violation. In our model the set of constraints is declared as consistent for each attribute definition. This means that an individual attribute definition must not specify conflicting validation rules.

Therefore, the set of all validation rules is consistent per type. However, since an attribute can be related to attribute definitions of different types, contradicting constraints are not completely excluded from our model.

It is important to note that attribute definitions and constraints are independent from the attributes of concrete content objects. On the one hand, this means that attribute definitions and constraints can exist even if no corresponding attribute exists for any content object. On the other hand, attribute definitions and constraints can also be defined even if some attributes and values violating the specified validation rules. The latter also means that the validity of a content object can change by defining integrity constraints without changing the content object itself.

Since constraints in our model do not force integrity of the structured information (i.e., attributes and values), they are called soft constraints. However, in Section 3.1.8 we explain how constraints can be defined more restrictive. Due to the fact that a typed content object not necessarily provides the attributes given by corresponding attribute definitions and that integrity is not forced by means of soft constraint, we refer to types as non-rigid.

In the following, we introduce some of the constraints in more detail. We refine them to more precise subtypes for purpose of illustration and in order to ensure consistency in our model. However, we keep the general model, that is, an attribute definition consists of an arbitrary number of constraints.

3.1.8. Constraints

The refined model as depicted in Figure 3.13 consist of two kinds of constraints. The multiplicity constraint allows to specify how many values an attribute should have. A lower bound and an upper bound can be defined, for example at-least-one (1..*), at-most-one (0..1), exactly-one (1..1).

The data type constraint validates if the data types of an attribute's values conform to a given type. For each basic data type as introduced in Section 3.1.5 a corresponding data type constraint is provided. For instance, for the basic data type string a corresponding string constraint can be specified. Besides the basic constraints for data types, additional constraints can be defined, such as the enumeration constraint (cf., Section 3.1.8.6).

Although the values of an attribute can differ regarding their data types, for example link and string (cf., Section 3.1.3), we consider constraining attributes to multiple different data types not to be beneficial. For this reason an attribute definition must not consists of more than one data type constraint. The same applies to the multiplicity constraint since it would not be reasonable if an attribute definition provides more than one multiplicity constraint. Therefore, at most one multiplicity constraint is allowed to be specified.

3.1.8.1. Default values

Besides the specification of validation rules for attributes and values, the data type constraint is used for the definition of defaults for attributes. Basically, defaults represent a set of anonymous values potentially having different data types. Anonymous means that these values

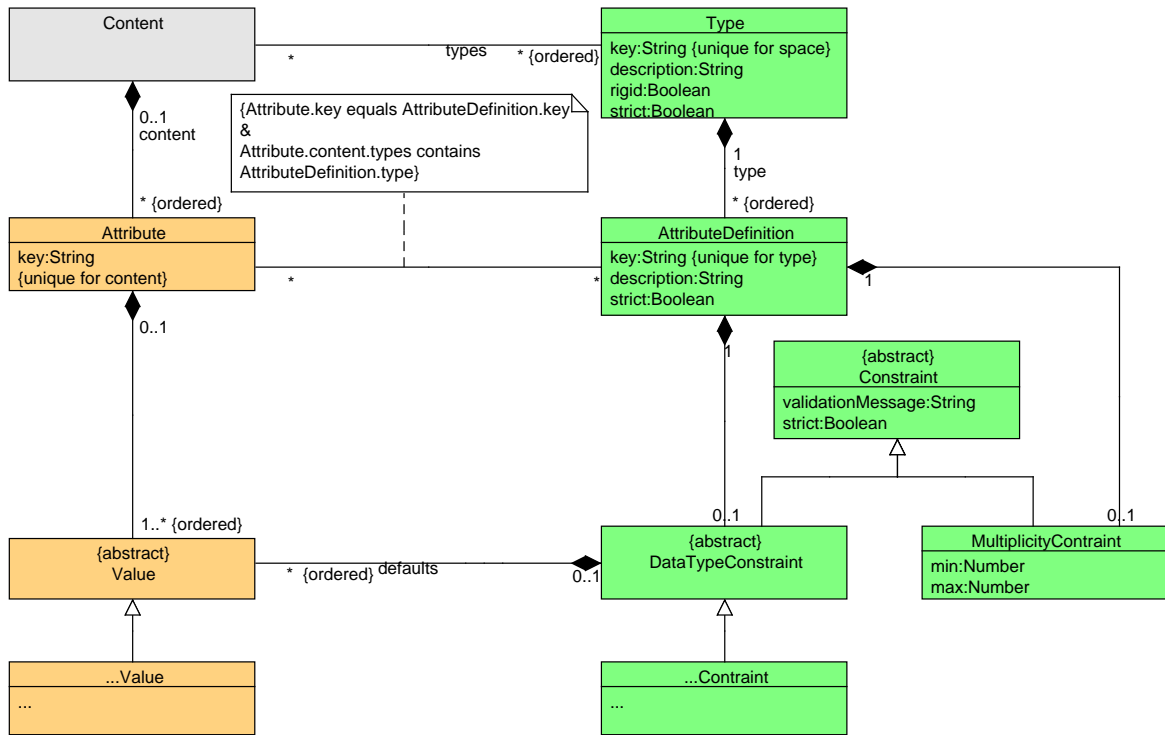


Figure 3.13.: Simplified model specifying types and integrity constraints.

are never related to any individual attribute. Therefore, the relation from value to attribute is declared optional (0..1) in our model as from now. For instance, defaults can be used while editing attributes in order to urge users to apply a specific value. By doing so, defaults facilitate the consistent usage of terms (cf., Section 3.3.1).

In order to ensure consistency in our model the data type of the defaults should match the data type constraint. For example, if the data type constraint is limited to links (cf., Section 3.1.8.8), the defaults should only be link values. Therefore, we decided that a data type-specific constraint is directly referencing only default values having the correct data type. For example, this means that a link constraint is directly pointing to link values for the definition of defaults. Alternatively, the defaults could also be bound to the attribute definition. The benefit of this alternative solution is that defaults can be specified independent of an individual data type constraint, that is, an attribute definition’s data type constraint can be changed without losing the current default values. But in this case it would be more difficult to ensure that the defaults correspond to a potentially given data type constraint. However, we decided to relate the defaults directly to the data type-specific constraints for purpose of illustration.

3.1.8.2. Validation model

Since attributes are basically independent of the constraints, individual attributes do not know about their current validation status. Therefore, for the discussion in this chapter the validation status of an attribute always has to be evaluated if it is needed. The simplest way

of applying constraints is on evaluating them always when displaying a content object. This means when a content object is requested by URL the related constraints are determined for each attribute and the validation status is calculated. For failing constraints then a validation message is displayed in the context of the corresponding attributes.

Figure 3.14 illustrates this simple validation model. A user requests a structured content object to be displayed (1), the platform determines the object and applies its constraints for purpose of validation (2). In case of violations, validation messages are shown (3b) in the context of the affected attributes, otherwise the content object is displayed as normal (3a).

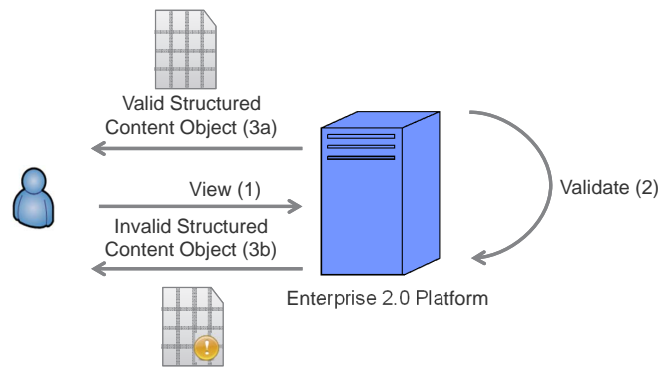


Figure 3.14.: Applying constraints when displaying a content object.

Since evaluating constraints only on demand potentially would slow down displaying content objects, the experience for users might get reduced while reading content, which is one of the most frequent use cases in Enterprise 2.0 platforms [AC10]. Another drawback of this solution is that searching for integrity violations would require to evaluate the constraints of all attributes in the entire platform on demand. Searching for invalid attributes in particular becomes necessary since a valid content object can change into invalid by defining an integrity constraint but without modifying the object itself. Therefore, it is important to provide search-based mechanisms in order to make the users aware of such implicit changes of the validity. Furthermore, users should not only see validation errors when reading a content object, they should also get informed about violations while editing it. Otherwise, they would need to save it in order to see if it is valid. In Chapter 4, we explain how to make the validation of content objects more efficient by persistently caching the current validation status within the attributes, how to explicitly search for constraint validations by using this cache, and how to check constraints while editing structured content.

Typically Enterprise 2.0 platforms provide a built-in validation model for content objects. For instance, it is most likely that the unstructured markup of a content object is validated by some rules, for example spell checking rules or limitations regarding the length of the content. Since the constraints as introduced in this chapter only refer to attributes and values, it is interesting to discuss how both models can be combined consistently. For example, if the models are integrated well the rule used for validating the built-in markup of a content object might be reused for validating the markup of a hypertext attribute. Since the integration of these models strongly depends on the given validation model of the underlying Enterprise 2.0 platform we return to this topic in Chapter 4.

3.1.8.3. Context-based constraints

The context of a constraint as proposed in our model is always given by a type and an attribute key. While this approach is more type-oriented, that is, limited to rules in the context of a type, and therefore limited in its flexibility of applying constraints, other more flexible models for the definition of constraints can be imagined. For example, the context in which a constraint is applied could also consist of exactly one attribute key as well as multiple types and spaces (cf., Figure 3.15). This attribute-centric solution would allow to share a constraint across different types and spaces. However, this flexibility would also cause more complexity in the management of constraints for the end-users. For example, it would be more difficult to communicate to the end-user which constraint applies in which context, in contrast to the type-centric approach, in which the context is always directly given by the type. In order to preserve simplicity we keep the type-centric approach in this thesis (cf., Section 2.1).

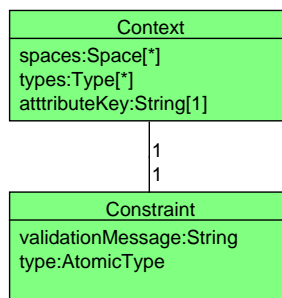


Figure 3.15.: Alternative validation model using a context for applying integrity rules.

3.1.8.4. Strict constraints

As depicted in Figure 3.13, constraints can be marked as strict. Strict means that a structured content object can only be saved if it does not contain invalid attributes or values. Thereby, users are forced to enter valid structured content by using strict constraints. Since this would contradict “an open editing philosophy that allows users to freely write and collaborate on web content without any restriction” [DIZ06] in Section 3.2.3 we explain how to support strict constraints without sacrificing the wiki way [LC01].

Furthermore, it is important to note that strict does not mean that the platform only contains valid structures. Since constraints can be defined and changed independent of individual attributes, an existing constraint can be set to strict even if some related attributes are violating this constraint afterwards. Considering the validation model introduced in Figure 3.14 this results in showing a validation message when matching content objects are displayed. Additionally, these content objects cannot be saved by users until all attributes and values are turned to valid ones or the constraint is changed to non-strict.

Types and attribute definitions can also be flagged as strict (cf., Figure 3.13). Basically, marking one of them as strict has the same meaning as described before, that is, content objects related to strict types or strict attribute definitions can only be saved if no corresponding

constraint is violated. But in detail their flags rather serve as defaults for the individual constraints. That means, individual constraints can be flagged as non-strict even if the corresponding attribute definition or type is declared as strict.

3.1.8.5. Rigid constraints

Types additionally can be marked as rigid (cf., Figure 3.13). A rigid type consists of strict constraints only, that is, a constraint cannot be declared as non-strict if its corresponding type is rigid. Furthermore, the set of attributes of a rigid typed content object cannot be changed, that is, attributes cannot be removed or added. Therefore, the set of attributes of a rigid typed content object at least contains the attributes defined in the context of the rigid type (i.e., the attributes which correspond to the type's attribute definitions). But it is not required that the attributes demanded by the attribute definitions have a value, because an attribute definition not necessarily provides a multiplicity constraint requiring at least one value. Since this is conflicting our model, in which each attribute consists of at least one value, these attributes are treated as suggestions (cf., Section 3.3.1).

If a rigid typed content object provides additional types (rigid or non-rigid), the attributes of these types are also included in that set of attributes. But even if all of these additional types are declared as non-rigid, this attribute set cannot be reduced or further extended for individual content objects, only by changing the type assignments. The limited adaptability of the attribute set only applies to individual, rigid typed content objects. This means that it is still possible to add or remove attribute definitions to or from rigid types.

Although the usage of rigid types seriously limits our approach in its flexibility (i.e., allowing users to freely assign and remove key-value pairs to or from content objects) it is crucial to solidify the state of the types at some point of time. For instance, it can be helpful to freeze the state of a set of typed pages for a certain period of time for purpose of analyzing their structured content or in order to generate a business report without being disturbed by changing attributes. However, in Section 3.2.3 we explain how this rigidity can be softened again in order to restore the possibility of flexible adaption and thus still to conform to the wiki way [LC01].

It is important to note that in our model explicitly marking a type as rigid expresses a certain concern of a business user and rigid typing is not caused by the technical limitation not being able to change the underlying information structures (or data model respectively) at runtime. Therefore, the dialog between the participants is facilitated regarding the composition of the information structures (i.e., types, attributes, and constraints). Basically, constraints in our model are not intended to hinder users in entering data they rather should guide them while managing information. In Section 4.2.3 we explain how users immediately benefit from constraints by providing advanced input controls for values.

In the following, we sketch the most important data type constraints with regard to this thesis. We focus on constraints having special properties whose purpose is not obvious. The other data type constraints namely string, number, hypertext, external link, and date are not described in detail, because they are all very similar, that is, only provide a set of default values with compatible data type. For instance, the defaults of a date constraint consist of

Moreover, internal link constraints can also be used to refine context information of link values. For instance, the context of a structured incoming link is implicitly given by the attribute the link value belongs to. In this case, the link constraint could specify a more meaningful name which is shown when viewing the incoming links of the target content object instead of showing the name of the corresponding attribute. Internal link constraints could also be used to provide additional restrictions regarding the inverse relationship between content objects. For instance, the inverse multiplicity could be defined with at-most-one (0..1). That is, a content object must not be referenced by more than one link value whose attribute name corresponds to attribute definition's name of the internal link constraint. However, we do not include advanced inverse properties, such as inverse role names or inverse multiplicity, in our model for reasons of simplicity. Internal link constraints are depicted in Figure 3.17.

A structured link constraint checks if an attribute only consists of structured links. Additionally, it inherits all properties from the link constraint as well as from the record constraint, that is, default links, default records, link target types, default types, record types, and default record types. Structured link constraints are depicted in Figure 3.17.

3.2. Emergent structures in Enterprise 2.0 platforms

In this section, we discuss how information structuring can be supported by Enterprise 2.0 platforms. First, in Section 3.2.1 we introduce different kinds of user participation in Enterprise 2.0 platforms. Based on this participation model, in Section 3.2.2 we describe the evolution of information structures provided by Hybrid Wikis. In Sections 3.2.3 and 3.2.4 we show the impact of Hybrid Wikis on existing Enterprise 2.0 services.

3.2.1. Participation model

In collaborative environments, such as in Enterprise 2.0 applications, users typically act in different roles. In Hybrid Wikis we distinguish three roles: visitors, authors, and tailors. The definition of these roles is taken from [DIZ06].

Visitors are users who read the actual content of an object. Their main activities within an Enterprise 2.0 application can be considered as view, search, navigate, and explore structured and unstructured information. Visitors use structured information primarily when viewing a set of similar object as part of a built-in view (cf., Section 3.4) or when searching for information by means of structured queries (cf., Section 3.2.4.1).

Authors are users who modify the actual content of an object. Their main activities within an Enterprise 2.0 application can be considered as edit, link, tag, discuss, and comment structured and unstructured information. Authors modify structured information only as part of the content objects, that is, they only add or remove attributes or assign types. Types, attribute definitions, and constraints are modified by tailors.

Tailors are users who modify the elements of the schema. Their main activities within an Enterprise 2.0 application can be considered as unify, constrain, and clean up data by maintaining types, attribute definitions, and constraints.

In [DIZ06] these three roles are used to differentiate between the users' activities in a wiki. In particular, persons acting as tailors configure and select validators, which are similar to integrity constraints in Hybrid Wikis. Therefore, it is most likely that tailors are also the users modifying the constraint's container (i.e., attribute definitions and types).

It is important to note that the same user can play different roles at different times. For instance, the same user who added an attribute to a content object can specify a related constraint later.

While visitors, authors, and tailors as introduced above represent human actors, these roles can also be played by an internal or external system accessing structured or unstructured information. This differentiation in particular becomes important in Section 4.2.7 when systems interact with Hybrid Wikis.

3.2.2. Evolution of emergent information structures

Hybrid Wikis provide a small set of structuring concepts, namely attributes, types, attribute definitions, and constraints (cf., Section 3.1). While attributes and type assignments enrich content objects with structured information on the data level, attribute definitions and constraints represent the constituents of the data's schema. That is, typed content objects can be considered as instances of these schema elements. However, in Hybrid Wikis the integrity of the data cannot be guaranteed by means of a schema, even when using rigid types or strict constraints (cf., Section 3.1.8). But due to the loose coupling of data and schema, both can be changed independently without being restricted by each other. In the following, the term data is used as a synonym for structured data or structured information respectively, subsuming structured content objects with attributes and type assignments. Furthermore, the term schema stands for types with attribute definitions and constraints. Therefore, on the schema level a type bundles a set of attributes with constraints, on the data level the assignment of a type expresses that a content object is an instance of this type.

Since in Hybrid Wikis the data's integrity cannot be ensured by means of the schema, we distinguish two kinds of schemata. The explicit schema is represented by types with attribute definitions and constraints, that is, all elements which are explicitly defined on the schema level. Whereas the implicit schema is based on the data. That means, the schema is implicitly given by the types (i.e., type assignments), attributes, and values of the structured content objects. Regarding the project example from Section 3.1.5.6 the content object *Max Leader* might be typed as *person*. In this case, the implicit schema would be represented by the types *project* and *person* connected by the link attribute *project lead*, even if both types do not specify any attribute definitions or constraints. Since the implicit schema is represented by structured data it can always be considered as the 'true' schema, whereas the explicit schema rather has a complementary character.

One goal of Hybrid Wikis is that the (explicit) schema incrementally emerges from the structured data (i.e., from the implicit schema). That is, emerging structures on the data level facilitate the evolution of the schema by giving recommendations which elements to explicitly define in the schema. For instance, if most of an attribute's values are dates it is suggested to define a date constraint for the related attribute definitions. Or the other way around, besides validating data the usage of types with attribute definitions and constraints also facilitates

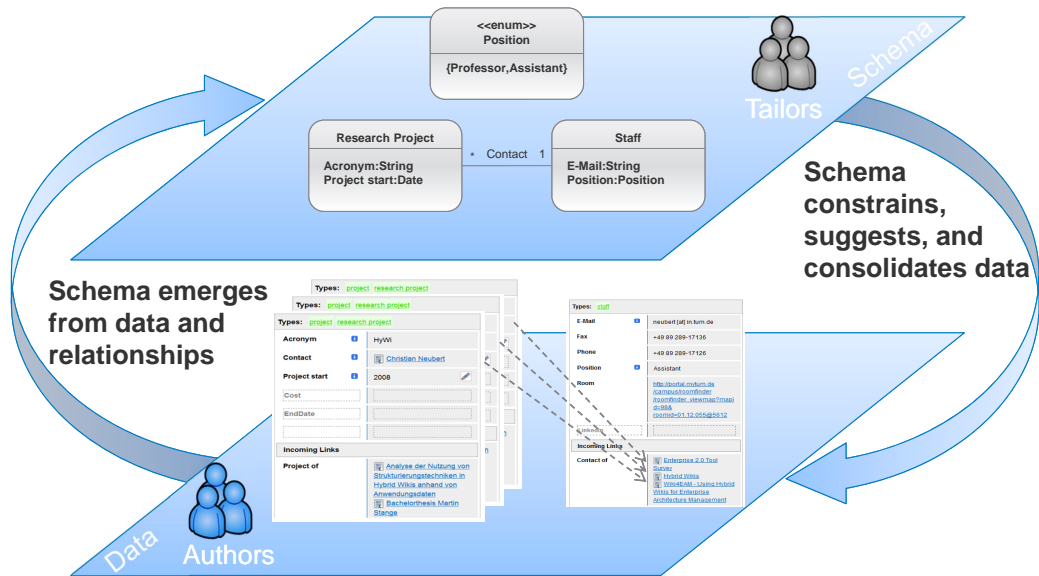


Figure 3.18.: Integrated data-driven information management and schema evolution via collaboration.

the consistent usage of data structures. For instance, if a date constraint is related to an attribute, users are urged to enter date values only. Additionally, schema elements are also used to provide structures by means of suggestions (cf., Section 3.3) and to increase the comfort for users when entering structured information (cf., Section 4.2.3). Figure 3.18 depicts Hybrid Wikis' two levels of structured elements, data and schema, and illustrates the interplay between them.

Hybrid Wikis support five stages of schema evolution (cf., Figure 3.19) in order to facilitate the bottom-up development of the schema. Bottom-up means that the schema incrementally emerges from the interplay of the individual structured content objects.

- Stage I: Key-values pairs (attributes) are specified for individual content objects.
- Stage II: Types are assigned to individual content objects.
- Stage III: Attributes are bound to types.
- Stage IV: Integrity constraints are defined for bound attributes.
- Stage V: Types and attribute definitions are declared as strict or rigid.

Each of these stages represents the degree of the schema's rigidity. The first two stages, Stage I and Stage II, are represented by structuring elements from the data level, the other stages consist of elements from the schema. For instance, if content objects are only structured by means of attributes and types the schema is only implicitly given by the data, that is, the content objects are lightly structured. In contrast, if most of the data is related to rigid types with attributes and constraints the schema is explicitly defined, that is, the content objects are heavily structured (cf., Figure 3.19). We assume (cf., Section 3.2.1) that authors primarily interact with elements from the data level (e.g., by structuring content objects) and tailors

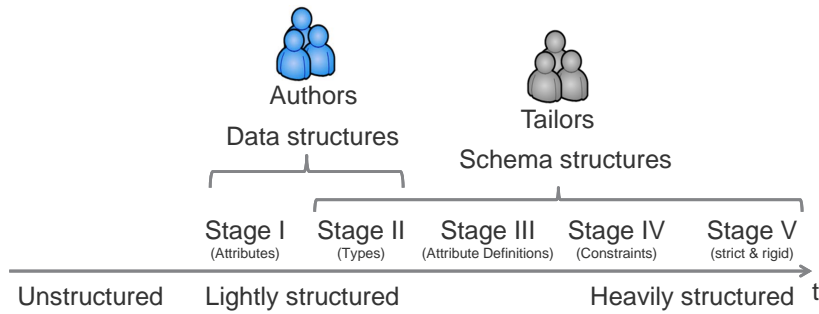


Figure 3.19.: Degrees of the structure's consolidation.

with elements from the schema (e.g., by managing constraints). Types (and type assignments) represent the connection between data and schema and therefore a connection between authors and tailors.

Although Hybrid Wikis facilitate the evolution of information structures and therefore information becomes more structured over time, they are not intended to replace unstructured content completely. Structured and unstructured information is rather complementary to each other. This also means that rigid structures can also be softened again. For instance, if types are declared to be non-rigid anymore, this is moving from Stage V back to Stage IV or removing types and attributes from an individual content object means moving from Stage II back to an unstructured state. However, Hybrid Wikis aim at constructing a well-defined schema by incrementally adapting information structures whereas the explicit schema emerges from the data. This way, the core information model continuously evolves and is hardening over time (cf., Figure 3.20).

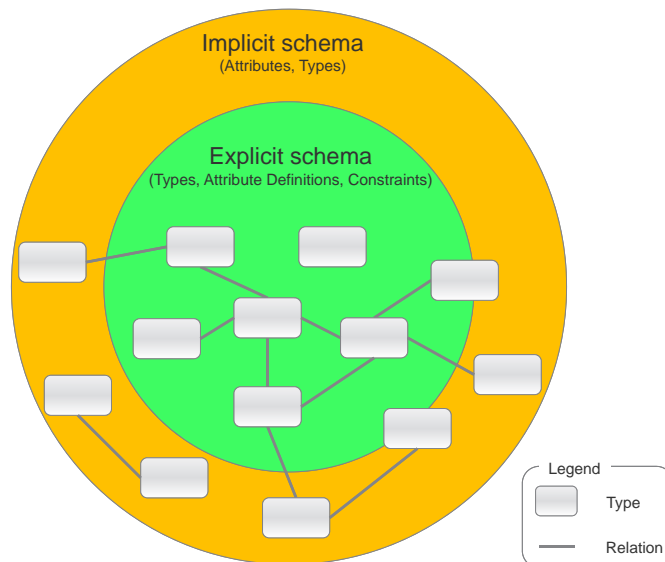


Figure 3.20.: Implicit and explicit schema in Hybrid Wikis.

3.2.3. Access control for structuring concepts

In the following, we discussed how the structuring concepts of Hybrid Wikis can be integrated into the access control model as introduced in Section 3.1.2.2. We differentiate between access control for information structures on the data level as well as for elements as part of the schema.

3.2.3.1. Access control for structuring individual content objects

As introduced in Section 3.1.2, content objects are protected from unauthorized access by means of access control permissions. Since attributes and the type assignments are fully integrated as parts of a content object these permissions are also applicable for them. That is, if a user is allowed to modify a content object she is also allowed to add, remove and modify attributes or to assign or remove types to it. The same applies to the read access. If a user is not allowed to read a content object, she does not see any of its attributes or types⁵. This means that in our model it is not possible to explicitly define access control lists for attributes or to specify who is allowed to assign a specific type. Which types are available for the assignment depends on their access rights and is discussed in Section 3.2.3.2. We consciously decided that an attribute only inherits the access rights from its owning content object, that is, it is not possible to define diverging permissions for it.

Especially in the context of an enterprise in many cases it is not required to exclude a set of attributes to be shown for a specific group of users due to reasons of confidentiality, it is rather required to create custom views showing only relevant information according to their business demands. How custom view can be created is introduced in Section 3.4.2. Furthermore, role-based hiding of attributes increases the overall complexity of Hybrid Wikis. For users it is more difficult to understand the different access right configurations, in particular if they are very fine-grained such as in case of different access rights for different attributes, even if they only differ for one individual content object. For instance, let p_I be a content object which can be viewed and edited by a user *Mayer*. Furthermore, let *budget* be an attribute having a value *100.000* which is not visible to *Mayer*. *Mayer* also might want to add *budget* to p_I since he is not aware of p_I already has such an attribute. Then an error message has to be shown saying that there already exists an attribute with the same key which is invisible for him. In this case, he is not even able to contact the attribute's owner (e.g., in order to ask about the attribute's current value or its purpose) since he cannot find out who has created the attribute. The main reason why view restrictions for attributes are avoided is that the collaboration on shared content object would become much more complicated and therefore would contradict the principles underlying Hybrid Wikis (cf., Section 2.1). Additionally, even if it would be possible to distinguish which user has created which attribute with which values in order to support duplicated keys (or values) this would seriously increase the overall complexity from a technical point of view.

In contrast, edit rights for individual attributes are rather comprehensible for users than

⁵This means that a user cannot reproduce if a type is assigned to any content objects. However, whether a type in general is visible depends on the access rights of the space the content object is related to (cf., Section 3.2.3.2).

restricted read access⁶. This is due to the fact that it is clear which attributes are available and who is the owner of an attribute. Therefore, different users can contact each other in case of doubts regarding the current state of an attribute, for example they can ask for changing the access rights. However, when providing versioning and awareness capabilities for information structures, changes to attributes can be tracked (cf., Section 3.2.4.3) and reverted (cf., Section 3.2.4.4), if necessary. In enterprises the degree of trust is higher when collaborating within an Enterprise 2.0 application [KGI08] than in open spaces such as in the Wikipedia, where in some cases so-called edit-wars emerge [VWD04]. Therefore, unauthorized modifications to individual attributes are rather unlikely in an enterprise context. For all these reasons Hybrid Wikis do not support edit access control for attributes.

Support of individual access control lists for attribute would seriously decrease the understandability of Hybrid Wikis from an end-user point and increase the complexity from technical perspective. Therefore, protecting attributes from unauthorized read or write access for any reasons (e.g., confidentiality) in Hybrid Wikis is only possible by means of defining a separate content object containing these attributes.

3.2.3.2. Access control for the schema

As described in Section 3.1.2, the access rights of a space primarily serve as defaults for the contained content objects, but can be specified for individual objects differently (cf., Figure 3.2). In case of types, the permissions of a space also apply to all its containing types, but cannot be overridden individually. That is, in our model types have derived access rights only, similar to the permissions of attributes. This means that users who are allowed to read object objects within a space are also allowed to see all contained types. The same way write access is derived. That means, users who are allowed to modify content objects within a space are allowed to modify all contained types. In particular, this also means that users with write access within a space can create new types or delete them. However, if a type can be assigned to an individual content object depends on the write permissions of this object.

The access control for attribute definitions and constraints works the same as for a type, that is, the access rights are inherited from the space. Since both are only indirectly associated with a space, the permissions are inherited from the related type or the related attribute definition respectively. This also means that the rigidity of constraints (or types and attribute definitions) can be changed by the editors (i.e., the authors) within a space, that is, constraints can be declared as non-rigid or non-strict, if needed (cf., Section 3.1.8).

Since types, attributes, and constraints are related to exactly one space (cf., Figure 3.1) they are only applicable to content objects within this space. This holds, even if a user is allowed to see (i.e., read) multiple spaces. For instance, let *Mayer* be a user who is allowed to read the two space *Management* and *Intranet*. Furthermore, let c_1 be a content object within *Management*, c_2 a content object in *Intranet*, and *contract* a type within *Management*. Then *contract* can only be applied to c_1 , even if *Mayer* is allowed to read *contract* in the context of c_2 . Of course, he could create a type *contract* within *Intranet* and assign it to c_2 . However, this would require to also duplicate the *contract*'s attribute definitions and constraints most

⁶For example, the *Budget* attribute of the *Development Project A* can only be changed by the responsible project manager.

probably resulting in diverging type structures over time. This issue could be addressed by providing a mechanism that allows to synchronize or to share (cf., Section 3.1.8.3) types across spaces but is currently not supported by Hybrid Wikis.

3.2.4. Integration of structuring concepts in Enterprise 2.0 services

Enterprise 2.0 applications provide a set of built-in services which are orthogonal operations on the content objects, such as tagging [GH05]. Orthogonal means that they can be applied to all content objects uniformly [BMN09]. In the following, we discuss the orthogonal applicability of these built-in services to the structuring concepts, namely attributes, types, attribute definitions, and constraints.

3.2.4.1. Searching and linking

Search services can be used to find content objects fulfilling specified criteria [Bi12]. In traditional Enterprise 2.0 platforms these criteria primarily target the full text (i.e., built-in markup) of content objects. That is, users can search for specific keywords within the unstructured content. In Hybrid Wikis these search capabilities are extended in order to additionally search for structured information. In particular, users can specify searches filtering content objects

- having a specific type,
- with a specific attribute, and
- providing a specific value.

These search filters can be combined by using boolean operators⁷. For example, it is possible to search for content objects having a specific set of types and additionally providing a specific set of key-value pairs. It is important to note that unstructured and structured queries can also be used in combination. For instance, it can be search for content objects with a specific type additionally containing a specific keyword in the full text of these typed objects. By doing so, structured and unstructured information can be accessed uniformly by means of queries. The ability to combine structured and unstructured queries is particularly important for “rich information repositories” [We09], such as Enterprise 2.0 platforms. Queries only related to the structured part of the contents we refer to as structured queries, queries referring to structured and unstructured elements are called hybrid queries. In particular, structured queries are used to produce dynamic views. Dynamic here means that the views are based on a set of data resulting from an executed query. In Section 3.4, we explain how custom views can be produced by using structured queries.

As introduced in Section 3.1.5.5 and 3.1.5.6, record and structured link values can be considered as anonymous objects in the context of its owning content object. Since both concepts provide attributes and types it is also required to include them when searching for structured information. Since these objects only exist in the context of its owning content object, that is, they do not have an own identity and do not provide a URL, they only can be accessed via

⁷Which combinations of search filters are currently supported by Hybrid Wikis is explained in Chapter 4.

the owning object. However, when searching for a specific type the search result potentially contains both kinds of typed objects, fully fledged content objects and anonymous objects, such a records and structured links. In such a case, accessing an anonymous object is only possible via the link of its owning content object.

As explained in Section 3.1.4, a type provides a description in order to give users the possibility to make a statement about its meaning. Therefore, the type can also be considered as content since the type's description, which is currently given as string, can also be modeled as rich text, similar to the built-in markup of a content object. Therefore, it is important to be able to search for keywords within this textual description in order to find types. For this reason it is required that types are treated as fully fledged object having an own identity and can be referenced by a URL. For instance, it can be helpful to link from a term within the full text of a content object to a type in order to explain the meaning of this term. In particular, this is useful when terms are overloaded and a more detailed description is required, such as in case of using the term service. The same also applies to attribute definitions (cf., Section 3.1.7), that is, it is possible to search for them and they can be referenced by URL. Constraints only exist in the context of its owning attribute definition. Since they do not provide a full text description they are not searchable and cannot be referenced by a URL.

Types, attribute definitions, and content objects are quite similar since all provide rich text, are searchable, and can be referenced by a URL. However, types and attribute definitions differ from content objects in so far that they cannot be structured by themselves, for example it is not possible to type a type again. We consciously avoided the structuring of structure since this would allow modeling on a meta level [St08] and most probably decrease the understandability of Hybrid Wikis from an end-user perspective.

As indicated in Section 3.1.8, it is possible to search for values violating one or more constraints. In particular, Hybrid Wikis allow to search for

- invalid attributes and
- objects containing any invalid attributes.

By combining different search filters it is possible to search for constraint violations within a specific context. For instance, it can be searched for typed content objects having invalid values for a specific attribute key (e.g., search for projects having more than one project lead). However, the user can only search for invalid attributes in general. It is not possible to define a query counting the attribute's number of values and comparing it to the bounds of a given multiplicity constraint. This is due to the fact that each attribute only provides a set of validation messages indicating the reason of the validation failure (cf., Chapter 4). The user has to manually check which constraint is responsible for the indicated violation. That means in the example the user has to manually check within the set of failing attributes whether the multiplicity constraint caused the violation or another constraint also applying to the attribute.

3.2.4.2. Tagging

Types and attribute definitions are similar to content objects since both can be searched, referenced by a URL, and provide a full text description. However, they cannot be structured

in order to avoid meta modeling (cf., Section 3.2.4.1). In [BMN09] tagging is described as “the process of collaboratively building a bottom-up categorization system”. This means by applying tags to types and attribute definitions it is possible to categorize them. For instance, tagging a type can be used for task organizing purposes [GH05]. In particular, when using “what-it-is” tags [GH05] tags can also be used to specify a type for a type, which is similar to meta modeling. However, since a tag in many cases is not used with type semantics (i.e., as a classifier), Hybrid Wikis support the tagging of types and attribute definitions. This is also due to the fact that it is still prevented to explicitly type a type or an attribute definition.

3.2.4.3. Versioning

Version management in [BMN09] is referred to as services “tracing the evolution of the content objects within their life-cycle”. That is, by means of versioning it can be traced which user has changed which parts of a content object at which time and for what reason. In Hybrid Wikis attributes and type assignments are recored as part of the version history of the content object itself. This means, it is additionally recorded which attributes and types changed (i.e., added, removed, updated) within the content object’s history.

Furthermore, the history also includes information about the validation status of a content object. That is, if the validation status of an object changes (e.g., by changing the value of an attribute) the validation message of the related constraint is recored. By doing so, it is reproducible at which point of time a content object turned into invalid and which constraint caused the violation. Changes of a content object’s validation status are even traced if the content object becomes invalid indirectly (i.e., without changing its types and attributes). For instance, a content object (i.e., one of its attributes) can turn into invalid if a constraint is specified separately. Versioning in this case serves the purpose of tracing a content object’s validation status. However, in most cases restoring a validation message is not useful since the causing constraint might have changed in the meantime. Therefore, the validation messages are always recalculated if restoring a content object from the version history in order to avoid inconsistencies.

Types and attribute definitions provide their own version history. The history of a type includes information about changes to its description, attribute definitions and tags. For an attribute definition it is traced which tags, constraints, and parts of the description have changed. The version history of a constraint is only part of its owning attribute definition.

3.2.4.4. Watching

Watching a content object means that users can register in order to receive a notification (e.g., via email) in case of changes to individual objects (cf., Section 3.2.4.3). Attributes and type assignments can only be watched indirectly by observing changes of state of content objects. This means that an attribute or a type cannot be watched individually. In contrast, it is possible to observe changes to types and attribute definitions directly.

3.3. Techniques facilitating information structuring

In this section, we explain how users interact with the structuring concepts introduced in Section 3.1. We focus on techniques that facilitate the structuring of information. In particular, we explain how suggestions for types, attributes, and data types on the instance level urge users to contribute structured content and how contextualized autocomplete mechanisms for terms and links provide means to develop a consistent schema in a bottom-up fashion. Additionally, we introduce a model describing transitions between unstructured and structured content that enables users to incrementally structure their information. Furthermore, we introduce consolidation mechanisms to unify data by means of the schema and show how structures defined for individual content objects can be transferred to the schema. Finally, we explain how built-in views reward users in providing information structures for example by providing better navigation capabilities through structured backlinks or tabular views listing the instances of a specific type.

3.3.1. Suggestions

One goal of Hybrid Wikis is to encourage users to structure information. Suggestions are a lightweight possibility to urge users to provide structures without forcing them. Additionally, suggestions in Hybrid Wikis help to

- reuse existing information structures,
- discover new information structures,
- indicate information structures for similar business demands, and
- unify terms and information structures.

In the following sections, we introduce suggestions for attribute keys, types, values, and constraints. Basically, the introduced suggestions are founded on an analysis of the structural similarity of content objects. For instance, two content objects have a high structural similarity if they have many of their attributes in common. In this case, if one object provides only one additional attribute this attribute (i.e., its key with an empty value field) is suggested to the user in the context of the other object.

Several variants of determining the similarity can be imagined. As in the previous example, the number of shared attribute key can indicate similarity, but also the number of types and shared attribute definitions can be used. Some of these variants are subsequently discussed, which we consider to be the most relevant. But the introduced suggestions are only based on the similarity of already existing structures, suggestions based on other information (e.g., based on an analysis of unstructured content [Ku08a]) are not considered in this thesis.

Due to reasons of usability (e.g., restricted space in the UI) as well as users' limited cognitive ability, only a small set of suggestions is offered to the users. Therefore, the similarity of content objects is used to determine a ranking within that set. The more similar a content object is to a content object in a specific context, the higher the probability that its structure is suggested to users in the context of the latter one. In this section, we sketch how the ranking of suggestions can be determined by proposing different priorities. In Chapter 4 it

is explained how suggestions can be calculated efficiently and how their ranking is actually implemented.

3.3.1.1. Attribute suggestions

We consider types to be the most important indicator for structural similarity. In the simplest case, exactly one type is assigned to a content object. Then the set of attributes used in combination with this type for other content objects is determined and the most frequent attribute keys are offered to the users as suggestions. For instance, let p_1 and p_2 be content objects both with type *project*. When p_2 provides an attribute *budget*, then *budget* is suggested in the context of p_1 , for example *budget* is suggested when p_1 is displayed.

In the case of multiple types per content object, not only the attribute frequency is considered but also if an attribute occurs for content objects having all or several of the types assigned. Such attributes are preferred in the list of suggestions. For example, let p_1 and p_2 be content objects each with type *project* and *research project*, and p_3 a content object with type *project*. Assuming p_2 provides an attribute *budget* and p_3 an attribute *status*, then both attributes, *budget* and *status*, are suggested in the context of p_1 , but the ranking of *budget* is higher since p_2 provides both types (*project* and *research project*), whereas p_3 only provides *project*.

If no types are present for a given content object, similar content objects are determined by taking into account only the attribute keys used. For instance, let p_1 and p_2 be content objects. p_1 provides the attributes *budget* and *status*, p_2 provides an attribute *budget*. Then *status* is suggested in the context of p_2 .

An attribute is more likely to be suggested if it occurs for content objects having all or several of the attributes currently assigned. Such an attribute is preferred in the list of suggestions, similar as in case of multiple types. For instance, let p_1 , p_2 , and p_3 be content objects. p_1 provides the attributes *budget*, *status*, *start*, p_2 the attributes *budget*, *status*, and p_3 the attributes *budget*, *end*. Then *start* is more likely to be suggested in the context of p_2 than *end*.

Attributes are also suggested for typed content objects with attributes related to attribute definitions and constraints. Since these definitions are part of the explicitly defined schema, they have the highest ranking of all attribute suggestions. For example, let p_1 and p_2 be content objects each with type *project*. p_2 provides an attribute *budget* and *project* an attribute definition *status*. Then it is more likely that *status* is suggested in the context of p_1 than *budget*. It is important to note that attribute suggestions originating from attribute definitions can be related to constraints. This means that a suggestion can become mandatory if using strict constraints or rigid types. For instance, if an attribute suggestion is related to a strict multiplicity constraint requiring exactly one value then a value has to be provided for the suggested attribute.

In [GH05] a classification for different kinds of tags is introduced, which are used throughout a collaborative tagging environment. One class of tags is classified as what-it-is. These tags are very similar to types with the difference that they are only plain keywords and cannot be used to specify any additional information, such as a description or attribute definitions. Since most of today's Enterprise 2.0 platforms support tagging of content objects tags are

also used in order to determine attribute suggestions. The determination works exactly as for types (single or multiple), but is based on tags instead. However, the ranking of attribute suggestions based on tags is lower than for those which are determined by the types since a tag not necessarily is intended to be used as a what-it-is tag. It is important to note that types can be considered as explicitly defined what-it-is tags. In Chapter 4 we introduce the concept of type tags as a lightweight connection between types and tags.

As mentioned before, attributes could also be suggested based on an analysis of the unstructured content. For instance, like in [Ku08a] the content could be analyzed using natural language processing methods in order to generate attribute suggestions. However, in Hybrid Wikis suggestions are only based on an analysis of existing information structures including tags.

The priority the ranking of the attribute suggestions is based on is given by:

1. Attribute definitions
2. Multiple types
3. Single types
4. Tags
5. Attributes

That is, an attribute is more likely to be suggested if it is related to an attribute definition and less probable if it is based on the analysis of the attribute keys only.

3.3.1.2. Data type suggestions

As explained in Section 3.1.5, each value provides a string representation. For instance, a hypertext value would be represented as a markup string including all formatting information, such as styles. In some cases the textual representation of a value with a specific data type can be interpreted as expected by another data type. In such cases the latter data type is suggested. For example, let *budget* be an attribute having a string value *100*. If the value's text is interpretable as a number, the data type number is suggested. Since all values have a textual representation, the data type string is not suggested. However, it is of course possible to change the data type to string (cf., Section 3.3.2.1).

3.3.1.3. Type suggestions

Types are suggested for a content object if it has all or some of the attributes other typed content objects provide. For instance, let p_1 and p_2 be content objects each with attributes *budget* and *status*, p_2 additionally provides the types *project* and *research project*. Then *project* and *research project* are suggested as types for p_1 .

Types are preferably suggested for a content object if its attributes corresponds to all or some of the attributes other typed content objects provide and attribute definitions are given for all or some of these attributes. For instance, let p_1 , p_2 , and p_3 be content objects each with attributes *budget* and *status*. p_2 additionally provides the type *project* and p_3 the type

research project. Furthermore, *budget* and *status* are attribute definitions of *research project*. Then both types, *project* and *research project*, are suggested as types for p_1 , but *research project* is preferred in the ranking of suggestions.

Types are suggested for a typed content object if other content objects with the same types provide additional types. For instance, let p_1 and p_2 be content objects each with types *project* and *research project*, p_2 additionally provides the types *current project* and *completed project*. Then *current project* and *completed project* are suggested as types for p_1 .

Furthermore, a type is suggested if a content object provides a tag and other content objects using this tag as a type instead. For example, let p_1 and p_2 be content objects. p_1 provides a tag *project*, p_2 a type *project*. Then for p_1 is suggested to use *project* as a type instead of using it as a tag.

A type is suggested if a content object has structured incoming links (cf., Section 3.1.5.2) and the key of the referencing link attribute is used in other content objects as a type. For instance, let p_1 , p_2 , and p_3 be content objects. p_1 is referencing p_2 by means of an attribute with key *project* and p_3 provides a type *project*. Then *project* it is suggested as type for p_2 .

The priority the ranking of the type suggestions is based on is given by:

1. Multiple types
2. Attribute definitions
3. Attributes
4. Incoming structured links
5. Tags

That is, a type is more likely to be suggested if other content objects provide additional types and less probable if it is using a plain tag instead of a type.

3.3.1.4. Autocompletion

In [LFZ09] autocomple is described as a method “which predicts a word or phrase that the user may type in based on the partial string the user has typed”. In this thesis, autocomple techniques are intended to guide the users while they enter structured information. Furthermore, autocompletion helps to use structures and terms consistently. Basically, three kinds of structuring elements are supported by autocompletion

- attribute keys,
- attribute values, and
- types.

Since potentially quite a lot of suggestions can be predicted based on a partial string, in particular in case of attribute values, the autocompletion results have to be determined by using a ranking. Subsequently, we sketch how autocompletion is supported in Hybrid Wikis, in Chapter 4 we explain how the ranking is determined from a technical point of view.

Autocompletion is provided for an attribute key if other content objects exist having attribute keys starting with the partial string the user has typed in. For instance, let p_1 and p_2 be content objects. p_2 provides an attribute with key *budget*. Then *budget* is suggested for p_1 as attribute key when typing *bu* while editing the key.

Autocompletion is provided for an attribute value if other content objects exist having attributes with the same keys and values starting with the partial string the user has typed in. For example, let p_1 and p_2 be content objects. p_2 provides an attribute with key *budget* and value *100.000*. Then *100.000* is suggested for p_1 as attribute value when typing *100.* while editing an attribute with key *budget*.

Autocompletion is provided for a type if other content objects exist having types starting with the partial string the user has typed in. For instance, let p_1 and p_2 be content objects. p_2 provides a type *project*. Then *project* is suggested for p_1 as type when typing *pro* while editing the types.

3.3.1.5. Constraint-based suggestions

Information specified by constraints (cf., Section 3.1.8) can also be utilized to provide suggestions. We consider suggestions for data types, values, types, and multiplicities to be relevant. The difference to the previously described kinds of suggestions is that constraint-based suggestions are user-defined and not generated by the system. In particular, constraint-based suggestions result from data type constraints, multiplicity constraints, and the constraint's defaults.

A data type is suggested if an attribute is related to an attribute definition with a data type constraint. For instance, let p_1 be a content object related to an attribute definition with key *manager* additionally specifying an internal link constraint (cf., Section 3.1.8.8). Then internal link is suggested as data type for the values of an attribute *manager* in p_1 .

Furthermore, additional information of the data type constraints is used to provide suggestions for values and types:

- Users are urged to contribute the values given by an enumeration constraint by preferably suggesting these values.
- If an internal link constraint with a specific set of link target types is specified, users are urged to provide only links with matching types by preferably suggesting matching links.
- Users are urged to use specific types for a record value (and for a structured link value accordingly) if a record constraint is restricted to a specific set of record types by preferably suggesting matching types.

For example, urging can be achieved by preferring constraint-based values and types in the suggestions' ranking or by using specific UI controls when editing values (e.g., present the enumeration values as a drop-down list).

A specific number of values is suggested to be filled out if an attribute is related to an attribute definition with a multiplicity constraint. For instance, let p_1 be a content object related to

an attribute definition with key *manager* additionally specifying a multiplicity constraint (cf., Section 3.1.8). Then it is suggested to provide the same number of values as specified by the constraint for an attribute *manager* in p_1 .

The defaults as introduced in Section 3.1.8 can also be considered as suggestions. We differentiate between defaults for values and types.

A default-based value is suggested if an attribute is related to an attribute definition (or to a constraint respectively) specifying a default value. For instance, let p_1 be a content object related to an attribute definition with key *status* additionally specifying a constraint with value *open* as default. Then *open* is suggested as value for an attribute *status* in p_1 . For example, when a user decides to create a new attribute from an attribute suggested in the UI (cf., Section 3.3.1.1) a default value can be initially filled as the attribute suggestion's value.

A default-based type is suggested if an attribute is related to an attribute definition (or to a constraint respectively) specifying a default type. For example, let p_1 be a content object related to an attribute definition with key *manager* additionally specifying a constraint with type *participation* as default (e.g., a structured link constraint, cf., Section 3.1.8.8). Then *participation* is suggested as type for an attribute *manager* in p_1 .

It is important to note that all constraint-based suggestions as well as attribute suggestions originating from attribute definitions with constraints (cf., Section 3.3.1.1) become mandatory if using strict constraints or rigid types (cf., Section 3.1.8). In this case, the same suggestions techniques can be applied, but with the difference that the users are rather forced to provide matching structures than they are urged only. However, wiki authors are free to declare constraints and types as non-strict or non-rigid any time.

3.3.1.6. Suggestions for attribute definitions and constraints

As described in Sections 3.3.1.5, 3.3.1.1, and 3.3.1.3, information from the explicitly defined schema, in particular from attribute definitions and constraints, can be used to provide suggestions on the data level. In order to facilitate the definition and evolution of information structures on the schema level attribute definitions and constraints are also suggested. By doing so, the schema and the data keep close together and the disparity of types and attributes is mitigated (cf., Section 3.1.4).

Basically, suggestions for the schema are derived from the concrete data instances (e.g., structured wiki pages). This means the determination of suggestions can be considered as guessing the schema. For instance, if a set of typed content objects all provide the same attribute⁸ with the same data type then it is reasonable to suggest a corresponding attribute definition with data type constraint. But even if only a few content objects in this exemplary set provide the same attribute, but differ regarding their values' data types (e.g., most of the values are links and only a few are plain strings), suggesting a data type constraint according to the data type of most of the values is still reasonable. Therefore, it is necessary to define thresholds for suggestions regarding the schema. That is, a schema suggestion is only provided if a certain threshold is exceeded. For instance, if a certain percentage of the instances' values are internal links then an internal link constraint is suggested, otherwise not. In the following, we intro-

⁸That means, they provide the same attribute key, but their values can differ.

duce suggestions for attribute definitions, data types, default values, and multiplicities. For all of them it is reasonable to define an individual threshold. However, we do not provide a concrete value for each of these thresholds in the following, we leave them as variables. This is also due to the fact that schema suggestions are currently being developed but not yet applied and evaluated in practice.

An attribute definition is suggested for type t if x percent of content objects typed with t specify the same attribute. For instance, let p_1 , p_2 , and p_3 be content objects, all typed with *project*. p_1 and p_2 provide an attribute with key *budget*. Furthermore, let 50 percent be the threshold for suggesting attribute definitions. Then *budget* is suggested as attribute definition for *project* since about 66 percent of *project*'s instances provide *budget*.

A data type constraint is suggested for an attribute definition *ad* if x percent of related attribute values provide the same data type. For example, let p_1 , p_2 , and p_3 be content objects, all are typed with *project* and provide an attribute *manager*. The data type of *manager* in p_1 is string, the data type of *manager* in p_2 and p_3 is link. Furthermore, let 50 percent be the threshold for suggesting data type constraints. Then an internal link constraint is suggested for the attribute definition *manager* since about 66 percent of its related attributes are of data type internal link.

Furthermore, additional type-specific elements are suggested for data type constraints:

- A link target type t is suggested for an internal link constraints if x percent of corresponding link values point to content objects with type t . For instance, let p_1 and p_2 be content objects, both with an attribute *manager* which is related to an attribute definition with an internal link constraint *ilc*. Furthermore, p_2 's attribute *manager* has a link value referencing a content object *Max Leader* with type *internal*. Let 50 percent be the threshold for suggesting link target types. Then *internal* is suggested as link target type for *ilc* since 50 percent of the link values referencing content objects with type *internal*.
- A record type t is suggested for a record constraints if x percent of the corresponding record values using t as record type. For example, let c_1 and c_2 be content objects, both with an attribute *address* which is related to an attribute definition with a record constraint *rc*. Furthermore, c_2 's attribute *address* has a record value with type *address*. Let 50 percent be the threshold for suggesting record types. Then *address* is suggested as record type for *rc* since 50 percent of the record values are typed with *address*.
- A value is suggested as part of an enumeration constraint if x percent of corresponding values are matching. That is, if an enumeration constraint already exists, this value is additionally suggested, otherwise a new enumeration constraint is suggested with exactly one value. For instance, let p_1 , p_2 , and p_3 be content objects. p_1 and p_2 provide the value *open* for an attribute *status*, p_3 provides *closed* as *status*. Then an enumeration constraint with value *open* is suggested since more than 50 percent of the values are the same.
- A value is suggested as default of a data type constraint if x percent of the related attributes provide the same value. For instance, let p_1 , p_2 , and p_3 be content objects. p_1 and p_2 provide an attribute *manager* each having a link value referencing a content object *Max Leader*, p_3 provides a string value *Müller* for attribute *manager*. Then, as

described above, an internal link constraint *ilc* is suggested. Additionally, the default link value *Max Leader* is suggested for *ilc* since about 66 percent of the related attributes provide *Max Leader* as link.

- A type is suggested as default link target type of an internal link constraint (the same applies to the default types of a record constraint and for the defaults of a structured link constraint) if x percent of corresponding link values point to content objects with type t . For example, let p_1 and p_2 be content objects, both with an attribute *manager* which is related to an attribute definition with an internal link constraint *ilc*. Furthermore, p_2 's attribute *manager* has a link value referencing a content object *Max Leader* with type *internal*. Let 50 percent be the threshold for suggesting default types. Then *internal* is suggested as default link target type for *ilc* since 50 percent of the link values referencing content objects with type *internal*.

A multiplicity constraint is suggested for an attribute definition *ad* if x percent of the related attributes provide the same number of values. In this case, the bounds (i.e., minimum and maximum) of the constraint are suggested. This means that the threshold must hold for both of these bounds. For example, let $p_1, p_2, p_3,$ and p_4 be content objects all typed with *project*. p_1 and p_2 provide an attribute *status* each having exactly one value. Furthermore, let 50 percent be the threshold for suggesting multiplicity constraints. Then a multiplicity constraint is suggested with lower bound 0 and upper bound 1 since 50 percent holds for both bounds as threshold. In case of an additional content object p_5 , typed with *project* and providing an attribute *status* with one value, the multiplicity constraint is not suggested since two of five attributes do not provide a value, that is, the threshold for the lower bound breaks. In this case, it would be reasonable to use 1 as the lower bound as well. However, in Chapter 4 we introduce a solution how this can be archived.

Suggestions for attribute definitions and constraints are intended to avoid that data and schema diverge over time. In particular, persons acting as tailors should become aware of the structural evolution and be encouraged to adapt the schema according to changing data. However, the previous introduced suggestions (i.e., in Section 3.3.1.6) can be considered as proposals since they are not applied in practice up to now.

3.3.2. Transitions

In the following, we first describe allowed transitions between structured attributes having different data types. Subsequently, we explain how unstructured and semi-structured information contained in the built-in markup of a content object can be transformed into attributes and objects, how attributes can be merged, and how attribute values can be transferred to objects. Finally, we sketch how these transformations and transitions can be combined in order to enable bulk operations.

3.3.2.1. Transitions between data types

All possible transitions for changing a value's data type to another one are depicted in Tables 3.1-3.3. As explained in Section 3.1.5, each value can be represented as a character sequence (i.e., as a string). For this reason, each value can be converted into a string in-

dependent of its actual data type (cf., first column of Table 3.1). For instance, changing a hypertext value into a string value would result in a textual representation of the plain hypertext including all formatting information, such as styles.

The transition between some of the data types is only possible if the value's string representation according to the source data type can be interpreted as required by the target. That means changing a data type to another one is only possible if the values' string representations are compatible. Data types which can be converted by interpretation are indicated in Tables 3.1-3.3. For instance, changing a string value to an external link is possible if the string can be interpreted as a URL, that is, if the string is matching the typical URL pattern (cf., Table 3.2). Another example is, transferring the string value *Mayer* to an internal link is allowed if the text *Mayer* conforms to the identifier of a content object, that is, the content object's name (or its URL) (cf., Table 3.3). In such a case, the text *Mayer* is replaced by an internal link to this content object (cf., Figure 3.21).

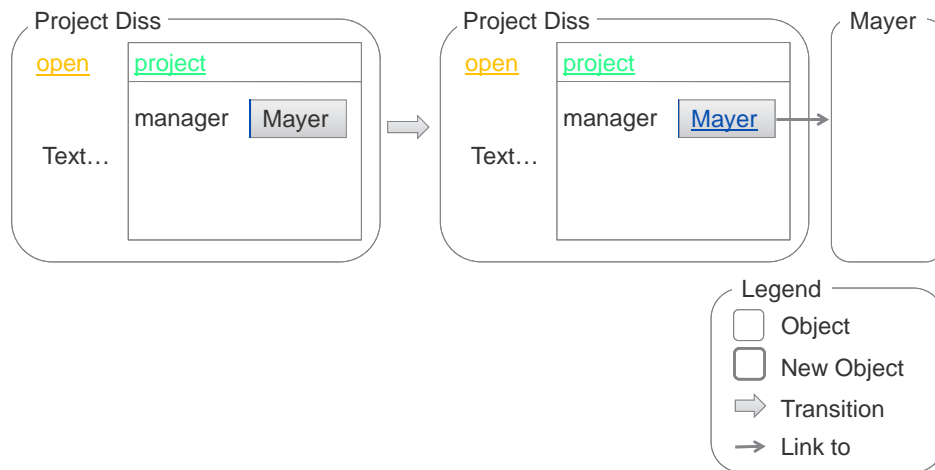


Figure 3.21.: Changing a value's data type from string to (internal) link.

In some cases a transition is only possible by providing additional information. For instance, a link additionally needs to be provided for the transition from record to structured link. But in some cases a transition is not possible without losing information. For instance, when changing a structured link to an internal link the additional structure (i.e., types and attributes) gets lost. However, in Section 3.3.2.4 we introduce a technique that allows to keep this information.

We say a transition t of a value v from a data type dt_1 to dt_2 is executable if v 's string representation according to dt_1 can be interpreted as required by dt_2 .

3.3.2.2. Transitions to attributes

The allowed transitions in order to transfer elements from the unstructured content to structured attributes are depicted in Tables 3.4-3.6. In particular, we consider the elements plain text, links, and hypertext to be relevant (cf., Figure 3.22 (c)) since they are most frequent and typical in unstructured content. Plain text differs from hypertext insofar that it does not

Table 3.1.: Transitions to data types String, Date, Number, and Hypertext.

From, To	String	Date	Number	Hypertext
String	-	if interpretable	if interpretable	yes
Date	as string	-	if interpretable	yes
Number	as string	if interpretable	-	yes
Hypertext	as string	if interpretable	if interpretable	-
Record	as string	no	no	no
ExternalLink	as string	no	no	yes
InternalLink	as string	no	no	yes
StructuredLink	as string	no	no	no

Table 3.2.: Transitions to data types Record and ExternalLink.

From, To	Record	ExternalLink
String	as string record value	if interpretable
Date	as date record value	no
Number	as number record value	no
Hypertext	as hypertext record value	if interpretable
Record	-	no
ExternalLink	as external link record value	-
InternalLink	as link record value	yes
StructuredLink	with losing the link	no

Table 3.3.: Transitions to data types InternalLink and StructuredLink.

From, To	InternalLink	StructuredLink
String	if interpretable	if interpretable or as string record value
Date	if interpretable	if interpretable or as date record value
Number	if interpretable	if interpretable or as number record value
Hypertext	if interpretable	if interpretable or as hypertext record value
Record	no	yes
ExternalLink	if interpretable	if interpretable or as external link record value
InternalLink	-	as link or as link record value
StructuredLink	with losing the records	-

contain any information regarding the layout. How a semi-structured table can be transferred to structured elements is explained in Section 3.3.2.5.

Additionally, we consider the transition between tags and attributes to be relevant (cf., Figure 3.22 (b)). For instance, it can be reasonable to transform a who-owns-it tag [GH05] to a

value of an attribute with key *owner* in order to explicitly make the purpose of this tag visible. However, a tag is a simple textual keyword and the transformation works the same way as transforming plain text from unstructured content to an attribute (cf., Table 3.4). Therefore, we do not explicitly explain the transformation of tags in the following.

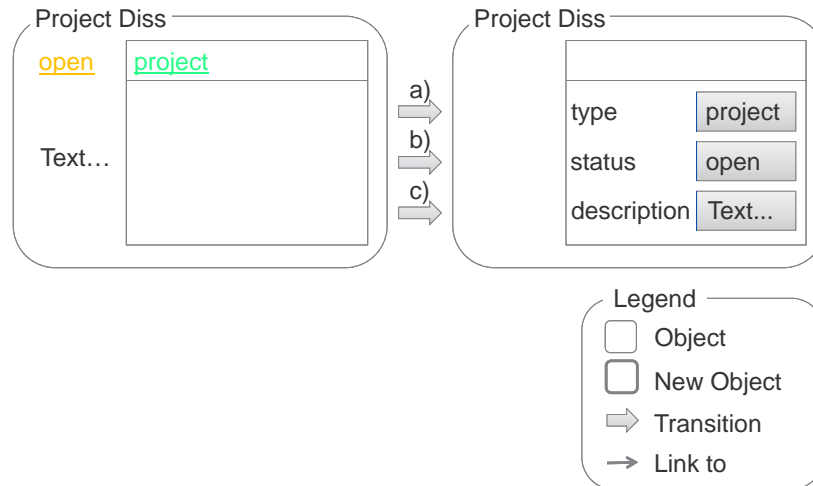


Figure 3.22.: Transitions from type to attribute (a), tag to attribute (b), and built-in markup to attribute.

Similar to tags, types can be transformed to an attribute's value (cf., Figure 3.22 (a)). For instance, in some cases a type indicates temporal dimensions (e.g., *completed project*) but does not provide any attribute definitions⁹ (e.g., the actual end date of the project). Then it can be reasonable to transform this type into a value of an attribute with for example key *status*. This has the advantage that constraints can be related to that attribute, such as an enumeration constraint specifying further status variations (e.g., *in preparation*, *started*) or a multiplicity constraints requiring that exactly one status is defined. Since the type's key is a simple string literal, the transformation (from type to attribute values) works the same way as transforming plain text from unstructured content to an attribute (cf., Table 3.4).

Since a structured attribute has to provide both, a key and a value, besides transforming the value (or a tag) to a matching data type, executing a transition additionally requires to provide a key. Furthermore, when structuring an unstructured element we decided that it is always completely transferred (i.e., deleted from the built-in content) to the structured attribute. This is different than additionally annotating unstructured information with meta data (i.e., merging data and meta data). This means, in Hybrid Wikis structured and unstructured information always remains clearly separated from each other. Transferring a tag or a type to a value is similar, both are completely replaced by an attribute after executing the transition. Deleting the source elements after a successfully executed transition helps to avoid information redundancies.

However, in case of removing elements from the built-in markup executing a transition potentially leads to fragmentary sentences, for example in case of removing the words *research*

⁹If the transferred type is related to any attribute definitions, the connection to the attributes would get lost after executing the transition.

project from the sentence *this page describes a research project*¹⁰. A possible solution would be to let the user decide whether to modify the source content or not. But this would require to always ask the user before executing a transition, which is potentially annoying and limits the agility of user interactions. Additionally, if users choose to keep the original text this would lead to information redundancies. Therefore, we decided to delete the transformed source element by default, even if users need to manually repair fragmented sentences in the built-in content in some cases. Another advantage of this decision is that users are encouraged to consciously distinguish between facts and descriptive contents.

In case of transforming text from the built-in content to an attribute, it is also conceivable to use the text as the attribute's key instead of as a value. For instance, the words *has a budget* from the sentence *has a budget of 200.000* can be used as attribute key. However, since an attribute can only be stored if at least one value is provided the value (e.g., 200.000) needs to be entered manually in this case. Furthermore, the user needs to manually repair the sentence after executing the transition, similar as described above.

Table 3.4.: Transitions from plain text.

To, From	String
String	yes
Date	if interpretable
Number	if interpretable
Hypertext	yes
Record	as string record value
ExternalLink	if interpretable
InternalLink	if interpretable
StructuredLink	if interpretable or as string record value

Table 3.5.: Transitions from hypertext.

To, From	Hypertext
String	as string
Date	if interpretable
Number	if interpretable
Hypertext	yes
Record	if interpretable or as hypertext record value
ExternalLink	if interpretable
InternalLink	if interpretable
StructuredLink	if interpretable or as hypertext record value

All previously introduced transitions can also be performed if the target attribute is already existing, instead of creating a new one. This means, if the source element is taken as a value it is appended to the target's existing values, if it is taken as the key the target's key is replaced.

Finally, transitions between attributes are relevant (cf., Figure 3.23). That means two attributes can be merged by appending the source values to the target. For instance, let *leader*

¹⁰In this example, the words *research project* could be used as an attribute's value with key *is a*.

Table 3.6.: Transitions from a link.

To, From	Link
String	as string
Date	no
Number	no
Hypertext	yes
Record	if interpretable or as link record value
ExternalLink	if interpretable
InternalLink	if interpretable
StructuredLink	if interpretable or as link record value

be the source attribute with value *Müller* and *manager* be the target with value *Mayer*. Then merging *leader* with *manager* means that the values' sequence of *manager* is *Mayer, Müller* after executing the transition (cf., Figure 3.23). Merging two attributes can be helpful for example in case of synonyms or for purpose of harmonizing keys in different languages.

We say a transition of a value v (or a tag t) to an attribute value of data type dt is executable if v 's (or t 's) string representation can be interpreted as required by dt .

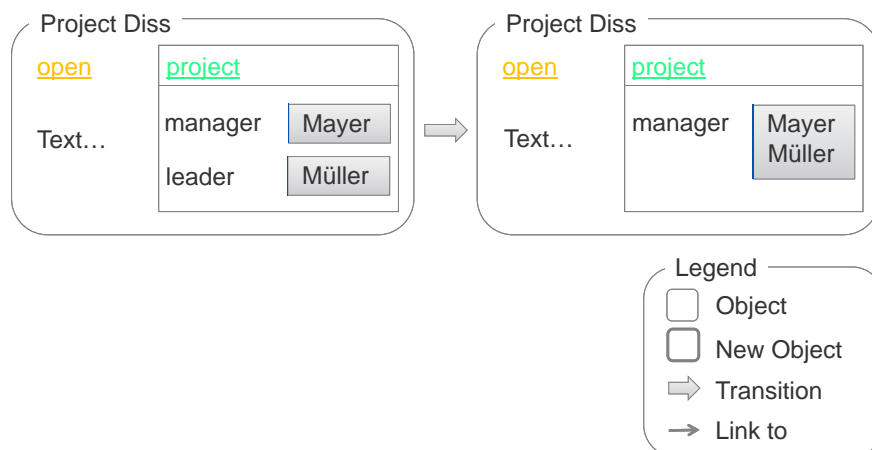


Figure 3.23.: Transition supporting the merge of two attributes.

3.3.2.3. Transitions to types

Unstructured content, tags, and attributes can be transformed in order to type a content object. On the instance level transforming one of these elements to a type means that a type is assigned to a content object using the element's string representation as the type's key. If no corresponding type exists, one is created before. Unstructured content mainly consist of hypertext including plain text and links. Even if hypertext and links can be represented in a textual manner it is most unlikely that this representation is used as the type's key. This is due to the fact that it potentially contains some layout information. Therefore, it is only reasonable to transform plain text into a type in most of the cases. For instance, it could be

reasonable to transform the two words *research project* from the built-in content into a type. However, in this case the problem of fragmented sentences occurs (cf., Section 3.3.2.2).

Additionally, it is possible to transform a tag into a type. For instance, it can be reasonable to transfer a what-it-is tag [GH05] to a type in order to make the purpose of the tag explicit, that is, the tag is used as a classifier.

In some cases it can even be beneficial to transform an attribute to a type. For instance, if using an attribute with key *type* its value is most probably a classifier. Then it is helpful to explicitly type the object with this value, for example in order to obtain better attribute suggestions (cf., Section 3.3.1). However, in some cases the value's string representation is not appropriate to be used as the key of a type (e.g., in case of a hypertext value).

3.3.2.4. Objectification of structured values

As described in Section 3.3.2.1, the data type of attributes can be changed. One possibility described is to change attributes to links, that is, if the string representation of a value can be interpreted as a reference to an existing content object. Instead of referencing an existing object when changing to data type link creating a new object is also a possibility. In this case, the attribute's value is merged into the newly created object and the link is pointing to this new object (cf., Figure 3.24). We differentiate three kinds of merging a value into the new object:

1. The value is taken as the name of the new object (cf., Figure 3.24).
2. The value is taken as the built-in markup of the new object.
3. The value is transferred to structured information within the new object.

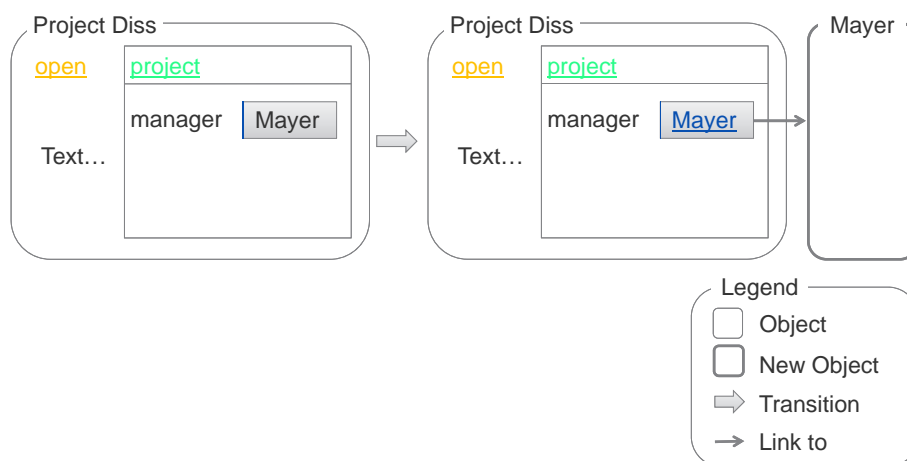


Figure 3.24.: Objectification of structured values.

By means of this technique an anonymous value is transferred into an fully fledged object having its own identity. Therefore, we call it the objectification of a value [Bi12]. How the value is merged into the new object depends on the attribute's data type.

In case of string, date, number, and external link, the value's string representation is taken as the name of the newly created object. A hypertext value is transferred to the built-in content of the new object since otherwise information would get lost (e.g., using a textual representation of the hypertext as the content object's name would only be reasonable if unnecessary formatting information is removed). When objectifying a record value its structure is transferred to corresponding types, attributes, and values in the new object. An internal link cannot be objectified since the new object would have the same name as the already referencing object (i.e., it is already a link).

The objectification of a structured link and a record are very similar, that is, types, attributes, and values of the link are transferred to the new object. In the first case (i.e., the objectification of a structured link), when replacing the original link with a link to the new structured object, the original referenced object would lose the information from which context it was referenced by, that is, the inverse link would get lost and the connection between the objects would get broken. One possibility to prevent this information loss, is the definition of an additional attribute in the originally referenced object also linking to the newly created object, whereas the key of the new attribute reflects the original context. For instance, let p_1 and *Max Leader* be content objects. p_1 provides an attribute *manager* which is a structured link to *Max Leader*. The structure of the link only consists of a type *participation*. When objectifying the link a new content object p_3 with type *participation* is created and the link from p_1 to *Max Leader* is replaced by an internal link to p_3 , in *Max Leader* additionally an attribute for example *participation* is specified referencing p_3 . However, in some cases the referenced object cannot be modified (e.g., due to access control restrictions). Therefore, an additional attribute cannot be specified in *Max Leader*. In such a case, the connection between p_1 and *Max Leader* can alternatively be preserved by specifying additional links in p_3 . That is, instead of replacing the links from p_1 to *Max Leader* by links from p_1 to p_3 and from *Max Leader* to p_3 , two additional links can be defined in p_3 , one from p_3 to p_1 and one from p_3 to *Max Leader*. In this case, the original link from p_1 to *Max Leader* is indirectly replaced by an incoming link from p_3 .

The objectification technique of structured links can also be applied when changing a value's data type from structured link to internal link (cf., Section 3.3.2.4). By doing so, changing the data types can be realized without losing structured information.

We say a transition t of a value v to an object o is executable if o can be created¹¹ successfully.

3.3.2.5. Objectification of semi-structured information

Besides transferring links, hypertext, and plain text from unstructured built-in content to structured attributes within the same object (cf., Section 3.3.2.2), a semi-structured table as part of the built-in content can be resolved by transforming it to newly created structured objects. The table is resolved by creating a new content object for each table row. The values of each row are transferred to values of structured attributes each of them having the corresponding column header as a key. Since this technique is transforming a table to fully

¹¹For example, an object cannot be created if the persistence layer raises an exception when storing it (e.g., due to reasons of concurrent write access).

fledged objects with own identity each this technique can be considered as the objectification of semi-structured information.

In many cases a table represents a set of similar objects having the same type. In case of a semi-structured (HTML) table, this type is only implicitly given or separately described in the full text (e.g., it is likely that the type of Table 3.7 is *project*). Therefore, a type (or multiple) can optionally be specified when objectifying a semi-structured table in Hybrid Wikis.

For instance, given the following table:

Table 3.7.: Table as part of the built-in content describing two projects with their budgets.

Project	Budget
Project A	10.000
Project B	100.000

When objectifying this table two content objects p_1 and p_2 are newly created. p_1 provides two attributes, *Project* with value *Project A* and *Budget* with value *10.000*, p_2 provides these two attributes accordingly. The data type of the attributes can be determined as introduced in Section 3.3.2.2. Since each content object must provide a unique name (cf., Section 3.1.2) it is also possible to explicitly designate one column (or more) to be used as the name of the new content objects. For instance, when using the *Project* column the values *Project A* and *Project B* are taken as the name of p_1 and p_2 . In this case, the newly created objects only provide the *Budget* attribute, otherwise redundant information would be given (cf., Figure 3.25).

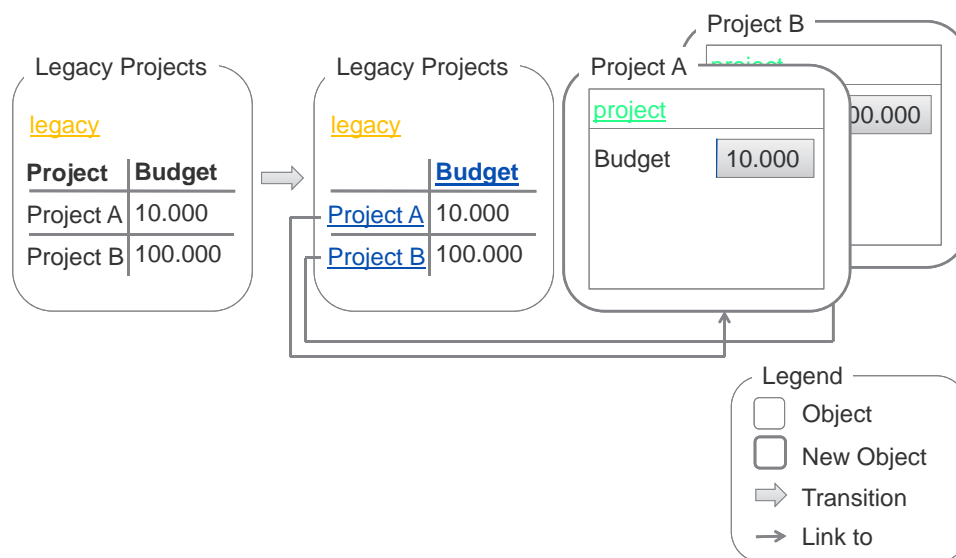


Figure 3.25.: Objectification of a semi-structured table.

But resolving a table to individual structured objects alone is not sufficient since otherwise the information originally contained in a single table would be scattered across multiple objects. Therefore, after executing the transition it is required to provide a view showing the same information as in the original semi-structured table, but using the newly created structured

objects instead (cf., Figure 3.25). In Section 3.4.2 we explain how this can be achieved by using embedded structured queries with different kinds of views.

We say a transition t of a value v to a set of objects $O=\{o_1, \dots, o_n\}$ is executable if each o in O can be created successfully.

3.3.2.6. Objectification of unstructured information

The technique of objectification can also be applied to elements of the unstructured content (cf., Figure 3.26). In particular, the elements plain text and hypertext are relevant to be explained. Plain text is used as the name of the new created content object and hypertext is transferred to the built-in content of the new object. In the latter case a name for the new object has to be specified manually. For instance, it is beneficial to objectify a paragraph (i.e., a hypertext section) if this piece of information should be referred to also in another context. For example, in case of textual describing multiple tasks on a wiki page one of these tasks can be objectified in order to associate it with the responsible person. Even if the objectification of unstructured information is not related to any of the introduced structuring concepts it is an important technique in order to create cross-linking knowledge.

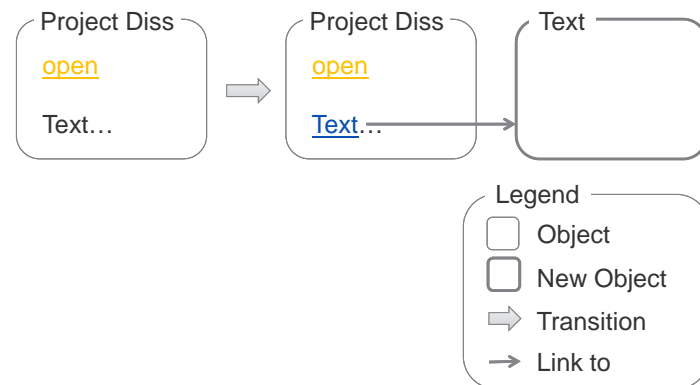


Figure 3.26.: Objectification of unstructured content.

3.3.2.7. Chaining transitions

In the previous sections we described different kinds of techniques that allow to transfer (structured or unstructured) values to structured attributes or structured objects. In some cases it can be helpful to execute multiple transitions at once. For instance, let t_1 be a transition from a value 100 (plain unstructured text) to a number (structured attribute) and t_2 be the objectification of a number. Then by chaining t_1 and t_2 100 can be transformed into an object in one step (i.e., a new object with name 100). In this case, t_1 and t_2 are dependent since t_2 can only be executed if t_1 's execution succeeded before.

We say a set T of (chained) transitions is executable if each t in T is executable individually and all dependent transitions can be executed successively. Chaining allows to flexibly combine different transitions in order to change the degree of structure for individual values or objects.

In particular, chaining of transitions allows to propagate changes within the schema to a set of related objects (cf., Section 3.3.3). For instance, a number constraint can be propagated to the related attributes if all attributes' data types are compatible with number.

3.3.3. Schema-based information consolidation

As discussed in Section 3.1, schema and data are only loosely connected in Hybrid Wikis. In this section, we explain how types, attribute definitions, and constraints can be utilized to consolidate and unify information structures on the data level. In particular, we show the consequences of applying, renaming, or deleting one of these schema elements for related objects and their information structures. We focus on how to propagate changes in the schema to the data. However, it is important to note that all introduced consolidation techniques are optional, that is, when changing the schema users can decide whether to consolidate the data or not.

3.3.3.1. Using constraints for data consolidation

The definition of a data type constraint implies that related attributes are validated correspondingly. However, the data types of these attributes can diverge from the data type of the constraint. Therefore, when defining a data type constraint it is possible to apply the constraint's data type to the related attributes. For this purpose the technique of chained transitions can be used (cf., Section 3.3.2.7). The type of a constraint can be propagated to related attributes if all data type transitions of the individual attributes are executable.

For instance, let p_1 and p_2 be content objects. p_1 provides a string attribute *manager*, p_2 provides a link attribute *manager*. Then when defining a link constraint lc which is related to *manager* the data type link can only be propagated if the data type transition for both attributes are executable, that is, in this case only if the *manager*'s value of p_1 can be interpreted as a link since the transition of the *manager*'s value of p_2 (link to link) is uncritical.

The propagation of data types can also be applied when replacing a data type constraint. For example, if the link constraint lc is replaced with a number constraint nc , the data type number can only be applied to *manager* if the link values of both attributes can be interpreted as numbers.

Similar to propagating data types, the technique of objectification can be applied to related attributes (cf., Sections 3.3.2.4 and 3.3.2.5). For this purpose the technique of chained transitions is used (cf., Section 3.3.2.7). For instance, let p_1 and p_2 be content objects. p_1 provides a string attribute *manager*, p_2 provides a link attribute *manager*, both attributes are related to the same data type constraint. Then the objectification of *manager* can only be propagated if the objectification is executable for both attributes, that is, in this case only if the *manager*'s value of p_1 can be objectified since the objectification of the *manager*'s value of p_2 (link to link) is uncritical.

It is also conceivable that a multiplicity constraint is propagated to related instances. However, applying a multiplicity constraint requires the definition or the deletion of values in some cases. For instance, if a multiplicity constraint with maximum 1 is applied to an attribute having

two values, one of them is required to be deleted. Therefore, this operation is not supported in Hybrid Wikis.

3.3.3.2. Using attribute definitions for data consolidation

Renaming the key of an attribute definition can be propagated to related attributes. For instance, let *manager* be an attribute of content object p_1 and ad_1 a related attribute definition (i.e., an attribute definition with key *manager*). Then when renaming ad_1 to *leader* p_1 's *manager* attribute is renamed accordingly (i.e., is renamed to *leader*). In case of p_1 already has an attribute *leader*, the values of *manager* are added to *leader*. Since attribute definitions are unique for a type, the key of an attribute definition can only be changed to an already existing key if both attribute definitions are merged together. Merging two attribute definitions means that all constraints of the source definition are added to the target, even if they contradict each other. Furthermore, the keys of related attributes of the source are renamed to the key of the target. For example, let *manager* and *leader* be two attribute definitions, *manager* provides a link constraint *lc* and *leader* provides a multiplicity constraint *mc*. Furthermore, let p_1 be a content object with attribute *manager* which is related to the attribute definition *manager*. Then *leader* provides both constraints, *lc* and *mc*, after merging the attribute definition *manager* into *leader*. Additionally, p_1 's *manager* attribute is renamed to *leader*. When merging two attribute definitions, the merged constraints can simultaneously be applied to related attributes of the target (cf., Section 3.3.3.1).

3.3.3.3. Using types for data consolidation

Since types can be shared among multiple content objects (cf., Section 3.1.4), renaming a type can be considered as changing the type of related content objects (i.e., the type's instances). Since types are unique within a space, the key of a type can only be changed to an already existing key when both types are merged together. Merging two types means that all attribute definitions and constraints of the source type are added to the target type, even if they contradict each other (cf., Section 3.3.3.2). Furthermore, the type of all source instances is changed to the target type. For example, let p_1 be a content object with type *project* and p_2 a content object with type *research project*. *project* provides an attribute definition *manager* with a link constraint *lc*, *research project* provides an attribute definition *area* with a multiplicity constraint *mc*. Then merging *project* into *research project* means that *research project* provides an attribute definition *manager* with both constraints *lc* and *mc* afterwards. Furthermore, both content objects, p_1 and p_2 , are of type *research project* after the merge. When deleting a type it can be decided whether to delete all related instances or only to remove the type assignment from related content objects.

Finally, the order of a type's attribute definitions can be applied to related instances. For example, let p_1 be a content object with type *project* and two attributes *budget* and *status*, ordered as *status*, *budget*. *project* provides two attribute definitions ordered as *budget*, *status*. Then applying the order of *project*'s attribute definitions to its instances results in changing p_1 's attribute order to *budget*, *status*.

3.3.4. Data-based schema adaption

Besides propagating changes in the schema to the data (cf., Section 3.3.3), it is also possible to apply structures from individual content objects to the schema. In this way, it is convenient for users to adopt suggestions regarding the schema (cf., Section 3.3.1.6).

If the key of an attribute changes (i.e., the key is renamed), it is possible to apply the rename operation to related attribute definitions. For instance, let p_1 be a content object having an attribute *budget* which is related to an attribute definition (i.e., with key *budget*). Then renaming of *budget* can be applied to the key of that definition.

The data type of an attribute's value can be passed to the schema as a corresponding data type constraint. For instance, let *budget* be an attribute with a number value. Then the data type number can be passed to related attribute definitions as a number constraint.

The number of an attribute's values can be passed to the schema as a corresponding multiplicity constraint. For instance, let *budget* be an attribute with one value. Then a multiplicity constraint can be specified for related attribute definitions requiring exactly one value for the attribute.

The attribute order of a typed content object can be applied to a type. For example, let p_1 be a content object with type *project* and two attributes ordered as *status*, *budget*. *project* provides two attribute definitions ordered as *budget*, *status*. Then applying the order of p_1 's attributes to the schema results in changing the order of *project*'s attribute definitions to *status*, *budget*.

In general, a rename operation can be expressed by executing two atomic operations, delete and (re)create. Therefore, changing the assignment of types can be interpreted as renaming a type, that is, removing a type from a content object and using another one instead. In such cases renaming of the removed type can be induced. For instance, let p_1 be a content object typed with *project*. When p_1 's assignment of *project* is replaced by *research project* the renaming of type *project* to *research project* can be induced. However, renaming a type cannot be applied automatically in this case. This is due to the fact that the user consciously wants to change an object's type assignment but leaving the types untouched. Therefore, the user explicitly needs to confirm such a rename operation.

The introduced techniques of schema adaption based on changes in the data can also be combined with the consolidation mechanisms introduced in Section 3.3.3. For instance, changing the key of an attribute definition resulting from renaming the key of an individual content object's attribute can be propagated to all other related attributes of the definition.

3.4. Views based on structured information

In order to access structured information Hybrid Wikis provide two kinds of views. Built-in views are provided by the system as defaults and custom views are tailored for user-specific information needs.

3.4.1. Built-in views

Built-in views primarily pursue two objectives with regard to the user:

- Facilitate information structuring.
- Create immediate benefit from structured information.

Since users are only willing to structure information if they obtain a benefit from it, that is, if the effort in structuring is less than the benefits, in Hybrid Wikis information structuring is stimulated by using UI elements users are familiar with, such as forms and spreadsheet-like tables. Our assumption is that they feel familiar with these representations and in consequence are less inhibited in contributing and manipulating structured data [MNS11].

In order to give users immediate access to the information structures a set of built-in views is provided mainly using forms and spreadsheet-like representations. Basically, four different built-in views can be utilized to represent the content (structured and unstructured) of individual objects.

- In the content view only the unstructured built-in markup is shown.
- The hybrid view is split in two equal parts, the left shows the unstructured built-in markup, the right the attributes and assigned types in form of a simple table.
- In the structured view attributes and type assignments are represented enlarged over the full size of the webpage.
- The link view shows structured information attached to outgoing links.

Furthermore, four views are provided to represent schema elements:

- The type table view shows all objects of a specific type in a table.
- In the attributes view a type's (derived) attributes and attribute definitions are shown.
- The types view shows all available structures within a space.
- The UML view shows all available structures within a space by means of a *Unified Modeling Language* (UML)-like class diagram.

3.4.1.1. Content view

The content view (cf., Figure 3.27) displays unstructured information (cf., Figure 3.27 (5)) with for example internal (cf., Figure 3.27 (3)) and external links (cf., Figure 3.27 (4)). Additionally, tags (cf., Figure 3.27(2)) and the object's name (cf., Figure 3.27 (6)) are shown. In order to append structured content users can activate the hybrid view (cf., Figure 3.27 (1)).

3.4.1.2. Hybrid view

The structured part of the hybrid view (cf., Figure 3.28) is presented as a box (cf., Figure 3.28 (1)) containing the list of key-value pairs at the right border of the webpage in two com-

3. Hybrid Wikis

Wikis » Project Example Diss » Hybrid Wikis - ... Last editor Christian Neubert - 2 minutes ago

View Details Attachments Versions Edit Watch Browse this Wiki Delete New Page Clone

Hybrid Wikis - Enterprise Collaboration and Information Management (6)

Tags: [social software](#) [hybrid wikis](#) [semantic](#) [wiki-based-approach](#) [wiki](#) [research](#) (2) (1) +

News (3)

20th of July 2011: The article *Hybrid Wikis: Empowering Users to Collaboratively Structure Information* was selected as the best paper at the International Conference on Software and Data Management 2011 (ICSOF 2011), Sevilla, Spain.

Objective (4)

Hybrid Wikis provide a lightweight semantic extension to the collaboration platform *Tricia* which comprises a traditional wiki as a core component. In contrast to more heavyweight approaches as semantic wikis, Hybrid Wikis do not focus on annotating wiki content with semantic information corresponding to a fixed ontology, which could be regarded as a top-down (i.e., ontology first) approach. In contrast, it allows a kind of ontology or datamodel to emerge from the content by enabling users to easily add structured content to any wiki page in the form of arbitrary named attributes or tags. Guidance is provided by suggesting terms that are frequently used by other users and the ability to derive new pages from existing ones. This bottom-up approach is complemented with the ability to establish certain constraints such as mandatory attributes or inheritance relationships between types. By applying these to stable parts of the schema that emerged over time further processing of the structured content - for example the generation of visualizations - becomes possible. (5)

Screencast

Watch this video to see Hybrid Wikis in action:

Figure 3.27.: The content view shows unstructured information, links, and tags of a wiki page.

partments. Attributes related to attribute definitions are shown in the first compartment (cf., Figure 3.28 (2)), attributes not related to a definition in the second (cf., Figure 3.28 (3)). Validation violations of attribute values are indicated accordingly (cf., Figure 3.27 (9)). The types are placed at the top of this box (cf., Figure 3.28 (4)). The appearance of the box is inspired by a kind of templates widely used in the MediaWiki software¹² – and thus in the Wikipedia [AL07] project. Since the box forms the structured part of the webpage, it can also be compared to the so-called fact box of Semantic MediaWiki [KVV06], that summarizes the facts being expressed by annotations in the text. However, in Hybrid Wikis neither does the attribute box reflect facts defined somewhere else nor is the selection of attributes specified by a template. Attributes and types represent the structured part of the content.

The box also contains attribute suggestions (cf., Section 3.3.1). Attribute suggestions are shown below the section containing the key-value pairs of an object. An attribute suggestion is presented in the same style as a normal key-value pair, having a key according to the suggestion, but showing an empty field for the value. By presenting an empty field for the value users are encouraged to fill out this field and information structuring is facilitated. Attribute suggestions originating from attribute definitions are displayed first (cf., Figure 3.28 (5)), other attribute suggestions below them (cf., Figure 3.28 (6)).

Additionally, the box always provides an empty row presented as an empty key-value pair, that is, providing an empty field for both, the key and the value (cf., Figure 3.28 (7)). Similar to attribute suggestions, users are encouraged to contribute new attributes and values by filling out these empty fields, like filling out the fields of a form.

Structured incoming links are the last part of the structured box. If the content object is referenced by a link value or a structured link (cf., Section 3.1.5), this is indicated in the box. In particular, the context of the referencing value is shown as a key (e.g., the value's attribute key) and the link to the referencing content object as value (cf., Figure 3.28 (8)). Representing a structured incoming link as a key-value pair integrated in the box also facilitates

¹²<http://www.mediawiki.org>; visited on January 3rd 2012.

Wikis » Project Example Diss » Hybrid Wikis - ... Last editor Christian Neubert - 1 minute ago

[View](#) [Details](#) [Attachments](#) [Versions](#) [Edit](#) [Watch](#) [Browse this Wiki](#) [Delete](#) [New Page](#) [Clone](#)

Hybrid Wikis - Enterprise Collaboration and Information Management (4) (1)

Tags: [hybrid wikis](#) [social software](#) [semantic](#) [wiki](#) [wiki-based-approach](#) [research](#)

News

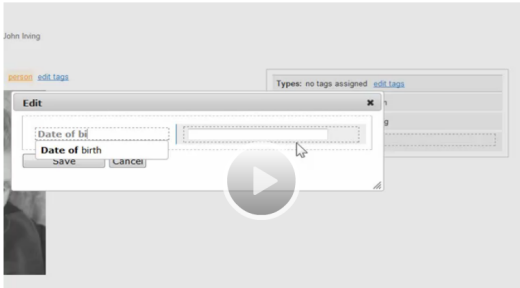
20th of July 2011: The article *Hybrid Wikis: Empowering Users to Collaboratively Structure Information* was selected as the best paper at the International Conference on Software and Data Management 2011 (ICSOF 2011), Sevilla, Spain.

Objective

Hybrid Wikis provide a lightweight semantic extension to the collaboration platform [Tricia](#) which comprises a traditional wiki as a core component. In contrast to more heavyweight approaches as semantic wikis, Hybrid Wikis do not focus on annotating wiki content with semantic information corresponding to a fixed ontology, which could be regarded as a top-down (i.e., ontology first) approach. In contrast, it allows a kind of ontology or datamodel to emerge from the content by enabling users to easily add structured content to any wiki page in the form of arbitrary named attributes or tags. Guidance is provided by suggesting terms that are frequently used by other users and the ability to derive new pages from existing ones. This bottom-up approach is complemented with the ability to establish certain constraints such as mandatory attributes or inheritance relationships between types. By applying these to stable parts of the schema that emerged over time further processing of the structured content - for example the generation of visualizations - becomes possible.

Screencast

Watch this video to see Hybrid Wikis in action:



Types: [project](#) [research project](#)

Area	<input type="checkbox"/>	Social Software	
	<input type="checkbox"/>	EAM	(2)
Start	<input type="checkbox"/>	2008	(9)
Status	<input type="checkbox"/>	active	

Acronym

HyWi

Contact (3)

[Christian Neubert](#)

Member

[Christian Neubert](#)
[Alexander Steinhoff](#)
[Dr. Christian M. Schweda](#)
[Prof. Dr. Florian Matthes](#)

End (5)

Manager (6)

Budget (7)

Sub-project (8)

Incoming Links

Project of (8)

[Thesis - Alexej Utz](#)
[Thesis - Katharina Rau](#)

Figure 3.28.: The hybrid view shows unstructured content on the left and structured content on the right.

the structuring of information since users are encouraged to enter additional key-value pairs in the box by following the same pattern. This is especially the case if a content object does not provide own attributes since then the structured box is even shown if the object is referenced by at least one structured link. In order to differentiate between keys of incoming structured links and simple attributes (i.e., the object's own attributes) within the structured box, the key of an incoming structured link is extended with the postfix *of*. For instance, let *Thesis - Alexej Utz* be a content object with attribute *Project* referencing a content object *Hybrid Wikis*. Then in the box of *Hybrid Wikis* the key *Project of* is shown having *Thesis - Alexej Utz* as link value (cf., Figure 3.28).

Incoming structured links are shown in a separate section of the box and can only be changed or deleted within the referencing content object. This box's section is similar to the incoming and outgoing links as displayed in the KiWi system¹³. Inverse links help to avoid redundancies, and thus inconsistencies, and to foster navigation in the system. Therefore, users are rewarded for structuring information.

Furthermore, the input fields for attribute names and values support autocompletion by dis-

¹³<http://www.kiwi-project.eu>; visited on January 2nd 2012.

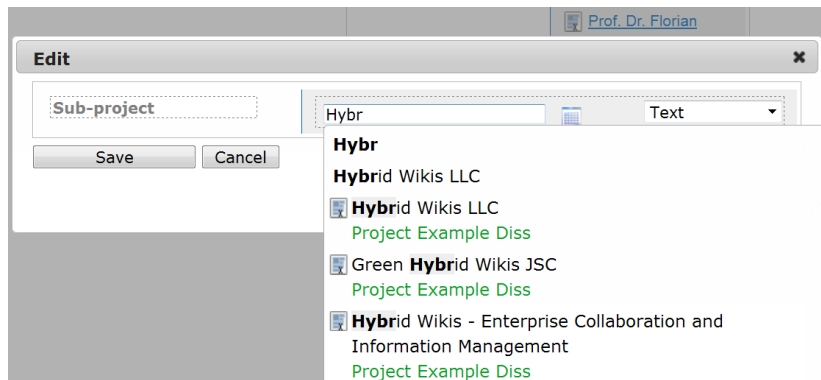


Figure 3.29.: Autocomplete suggestions for attribute values.

playing suggestions as the user is typing (cf., Section 3.3.1.4). When an attribute value is typed, the respective attribute name is factored in to improve the quality of the suggestions (cf., Figure 3.29). This makes it comfortable to contribute structured content and additionally fosters consistent usage of terms. The same applies to types, that is, the user is assisted by autocompletion when assigning a type.

3.4.1.3. Structured view

If using the hybrid view in case of a heavily structured content object (i.e., an object mainly consisting of structured content), splitting the webpage would be a waste of space and additionally limits the readability of the structured content. In such a case the representation can be changed to the structured view (cf., Figure 3.30), that is, enlarging the structured box with attributes (cf., Figure 3.30 (1)) and types (cf., Figure 3.30 (2)) to the full size of the webpage. Built-in attributes (e.g., built-in content) are integrated in the enlarged box as key-value pairs (cf., Figure 3.30 (3)).

3.4.1.4. Link view

The link view shows structured information (i.e., attributes and types) attached to (outgoing) links in a table (cf., Figure 3.31). The link target (cf., Figure 3.31 (1)), its context (i.e., the attribute key, cf., Figure 3.31 (2)), its types (cf., Figure 3.31 (3)), and its attributes (cf., Figure 3.31 (4)) are displayed as a table column each. The link's source is not displayed since it is the same for all links. The attribute keys are sorted in descending order by the frequency of provided values in order to indicate the degree of the attributes' importance (cf., Figure 3.31 (5)). Each link is represented by exactly one table row. Above of the table the links' types are presented in a tag cloud in order to filter for links with a specific type (cf., Figure 3.31 (6)). This way, attributes of other types can be hidden.

These four different built-in views can be activated for content objects individually. Per default the content view is activated. However, if at least one attribute or type is assigned, the content object is presented in the hybrid view. The hybrid view is also taken as default if at least

Wikis » Project Example Diss » Hybrid Wikis -... Last editor [Christian Neubert](#) - 1 minute ago

View Details Attachments Versions Edit Watch Browse this Wiki Delete New Page Clone

Hybrid Wikis - Enterprise Collaboration and Information Management

Tags	social softwa semantic wiki wiki-based-approach research
Content (3)	<p>News</p> <p>20th of July 2011: The article Hybrid Wikis: Empowering Users to Collaboratively Structure Information was selected as the best paper at the International Conference on Software and Data Management 2011 (ICSOFT 2011), Sevilla, Spain. ...</p>
Type Tags	project research project (2)
Area (1)	Social Software EAM
Start	2008
Status	active

Figure 3.30.: Attributes, types, and built-in content presented in the structured view enlarged to the full size of the webpage.

Structured Links Last editor [Max Mustermann](#) - 7 minutes ago

View Details Attachments Versions Structured Links Edit Watch Browse this Wiki Delete New Page Clone

Structured Links from Hybrid Wikis - Enterprise Collaboration and Information Management

[participation](#) (6)

To (1)	Name (2)	Types (3)	Since(4) (5)	Bonus(1) (4)	Until(1)
Christian Neubert	'Contact'	no tags assigned			
Christian Neubert	'Members'	participation	01.09.2008	1.000.00 €	
Dr. Christian M. Schweda	'Members'	participation	01.09.2008		01.09.2011
Alexander Steinhoff	'Members'	no tags assigned	01.06.2009		
Prof. Dr. Florian Matthes	'Members'	no tags assigned	01.09.2008		

Figure 3.31.: The structured link view allows to attach attributes and types to hyperlinks.

one incoming link is structured. Users can change to the structured view and the link view manually.

3.4.1.5. Type table view

Besides the representation of a single content object, a set of content objects having the same type can be shown in a built-in tabular view. Since this table contains all instances of a particular type we refer to it as type table view or type table respectively. Each table row corresponds to one content object and its structure (cf., Figure 3.32 (1)), whereas the first column shows a link to the object (i.e., the name of the object as the link's text). The other columns represent the attributes' keys sorted in descending order by the frequency of provided values in order to indicate the degree of the attributes' importance (cf., Figure 3.32

3. Hybrid Wikis

(2)). Additionally, it is indicated which attributes originate from the data only and which are statically bound to the type by means of an attribute definition (cf., Figure 3.32 (3)). Similar to the hybrid view, constraint violations are highlighted (cf., Figure 3.32 (7)). New instance of the type underlying the table can easily be added (cf., Figure 3.32 (4)). The type's description is shown above of the table (cf., Figure 3.32 (5)). The tags of a type can for example be used to generate an enterprise glossary (cf., Figure 3.32 (6)).

Wikis » Project Example Diss » Types » Instances

Instances Details Attributes Watch New Type Edit Type Delete Type New Attribute Definition

Entities with Type Tag project in Project Example Diss

Tags: glossary_entry glossary **(6)**

A project in business and science is typically defined as a collaborative enterprise, frequently involving research or design, that is carefully planned to achieve a particular aim. Projects can be further defined as temporary rather than permanent social systems that are constituted by teams within or across organizations to accomplish particular tasks under time constraints. [Wikipedia] **(5)**

Showing 1 to 7 of 7 entries Search: **(2)** First Previous 1 Next Last Show 10 entries

	Start (7) (3)	Status (7)	End (6)	Manager (6)	Area (4)	Contact (4)	Member (4)	Budget (3)	Sub-project (2)	Acronym (1)
Calculation	06.12.2011	active	06.05.2012	Hans Mayer						
Development Project Diss (1)	06.06.2011	active	06.06.2012	Max Mustermann	Social Software	Don Jose	Hans Mayer Jane Doe Max Mustermann	200.000	Calculation Project Documentation	
Feed the whales	01.02.2011	completed	20.04.2011	Jane Doe				1.000		
First Migration Project	10.06.2012	planned	15.11.2012	Jean Dupont	Social Software	Ola Nordmann		50.000		
Hybrid Wikis - Enterprise Collaboration and Information Management (7)	2008	active			Social Software EAM	Christian Neubert	Christian Neubert Alexander Steinhoff			HyWi

(4)

Figure 3.32.: The type table view shows all instances of a type.

Users can access the type table view by clicking on a type, for example in the hybrid view (cf., Figure 3.28 (4)). Thereby, authors immediately benefit from typing an object since they obtain a tabular overview of all objects having the same type and they can manage (i.e., create, edit, delete) attributes like in a spreadsheet using in-place editing (cf., Section 3.4.4). Tailors can use the type table to manage (i.e., create, edit, delete) attribute definitions and constraints. The indication of the frequency of provided values helps to identify candidates for attribute definitions, that is, frequently used attributes that might be bound to the type by means of an attribute definition. Furthermore, tailors can reflect the quality of their decisions regarding the schema by checking the usage of currently bound attributes. For example, if a tailor introduces a new attribute definition which is only rarely used by the authors after a while (i.e., only a few content objects provide a value for this attribute) this may indicate that the 'invented' attribute is rather less suitable. By means of the type table view tailors and authors can enter into dialogue which helps to cure the so-called ivory tower syndrome [Bu10], that

is, the creation of ‘invented’ information models unsuitable for persons that provide the data is avoided.

3.4.1.6. Attributes view

By means of the attributes view users can manage the attributes of a type. Figure 3.33 depicts the attributes of the type *project*. It is indicated which attributes are related to an attribute definition (cf., Figure 3.33 (1)). By default the attribute keys are sorted in descending order by the frequency of provided values (cf., Figure 3.33 (2)). Attribute keys can be changed (cf., Figure 3.33 (3)) in order to support schema-based consolidation techniques (cf., Section 3.3.3), in this case renaming of an attribute can be applied to related instances. For each attribute its defined constraints (cf., Figure 3.33 (4)) and the suggested constraints (cf., Figure 3.33 (5)) are shown. In this example, no constraint is defined for the attribute *Manager*, but since 100 percent of the current values are links to objects typed with *internal* and *staff* a corresponding link constraint is suggested. This way, tailors are urged to specify constraints based on instance data (cf., Section 3.3.1.6).

Wikis » Project Example Diss » Types » Instances » Attributes ?

Instances New Attribute Definition

Attributes with type project (7) in space Project Example Diss

Attribute	Name	Frequency (2)	Type	Type Usage	Multiplicity
Start (7)	Start	7/7 (100 %)		Text (100%)	
End (6)	End	6/7 (85 %)		Text (100%)	
Manager (6)	Manager	6/7 (85 %)		internal , staff (100%) (5)	
Status (6):Enumeration (1)	Status (3)	6/7 (85 %)	Enumeration planned active completed canceled (4)	Text (100%)	not selected
Budget (5)	Budget	5/7 (71 %)		Text (100%)	
Area (4):Text	Area	4/7 (57 %)	Text	Text (100%)	not selected
Contact (4)	Contact	4/7 (57 %)		external , staff (75%) Link (25%)	
Members (4)	Members	4/7 (57 %)		internal , staff (75%) Link (25%)	
Sub-project (2)	Sub-project	2/7 (28 %)		project (83%) Text (16%)	
Acronym (1)	Acronym	1/7 (14 %)		Text (100%)	

Figure 3.33.: The attributes view shows all attributes of a type.

3.4.1.7. Types view

In order to get an overview of all available structures within an individual space (e.g., wiki) the so-called types view is provided built-in (cf., Figure 3.34). This view shows all types (cf., Figure 3.34 (1)) of a space with their descriptions (cf., Figure 3.34 (2)), attributes (cf., Figure 3.34 (3)), and constraints (cf., Figure 3.34 (4)) in a table.

Each table row represents one type and the first column shows a link to this type. In the second

Type Tag	Type Definition
customer (2)	A customer (also known as a client , buyer , or purchaser) is the recipient of a good, service, product, or idea, obtained from a seller, vendor, or supplier for a monetary or other valuable consideration. [Wikipedia] Address (2):Text Customer Number (2):Number Projects (2): project (7)
external (3) (1)	An external is not employed in the company. (2) Company (3): customer (2) (4) Decision Maker (3) EMail (3) Firstname (3) (3) Lastname (3)
internal (4)	An internal is employed in the company.

Figure 3.34.: The types view shows the all available structures within a space.

column the types' attributes and constraints are shown sorted by the frequency of provided values, whereas attributes related to a definition are shown above simple attributes.

3.4.1.8. UML view

In [Ut11] the author introduces web-based visualizations of models created with Hybrid Wikis. The solution is inspired by UML class diagrams and considers information from the explicitly defined schema (i.e., types, attributes definitions, and constraints) as well as from the implicit schema which is derived from instance data only (i.e., structured content objects with attributes and type assignments).

The author developed a mechanism to combine these two schemes in a single *Eclipse Modeling Framework* (EMF)¹⁴ model. In this model, explicitly defined elements (e.g., attribute definitions) hide elements which are derived from the data, also in cases if the derived information is more specific than the defined one. For instance, let p be an content object with attribute *budget* having a link value. Furthermore, let *budget* be related to an attribute definition (i.e., with key *budget*) having a number data type constraint. Then in the combined model the more specific data type (link) is hidden by the defined one (number) and *budget* is represented as an attribute with data type number. However, in cases if no corresponding information is defined the derived element (e.g., the derived multiplicity) is used in the model.

Figure 3.35 depicts the extended UML class diagram representing the structures within a wiki used for project management (cf., Section 2.2). Defined elements are presented in green, derived elements in blue. Strict elements show an exclamation mark. For classes and properties the number of instances is indicated. The number of invalid (orange) and wrong (red) instances is indicated separately.

The developed solution helps to get an overview of existing information structures within a space. Additionally, the UML view facilitates the navigation within the structures since some

¹⁴<http://www.eclipse.org/modeling/emf>; visited on April 05th 2012.

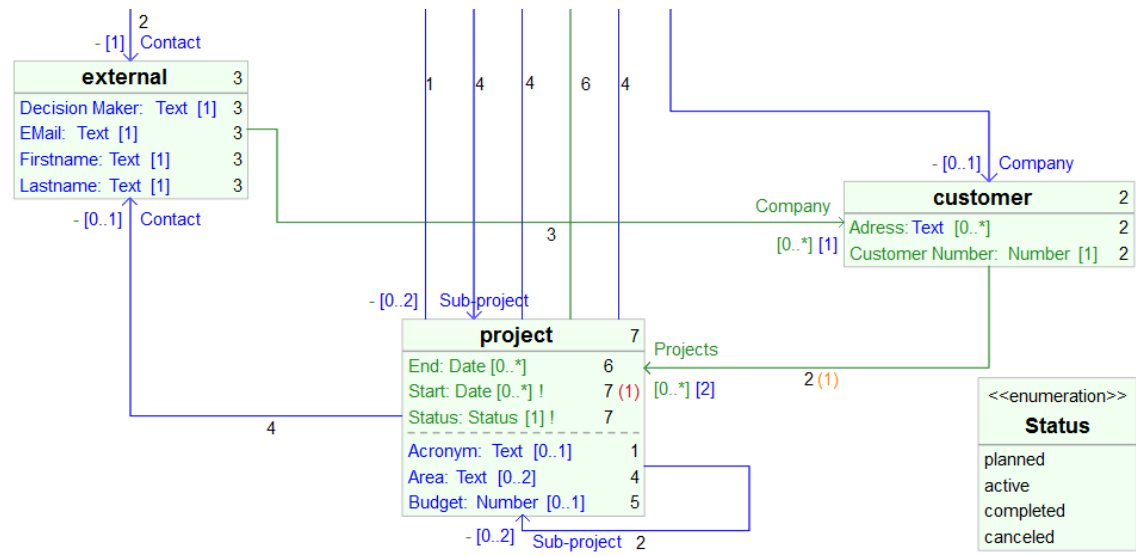


Figure 3.35.: Visualizing emergent information structures according to [Ut11].

elements are ‘clickable’. For instance, the type table view is shown when clicking on a class in the diagram. In Chapter 6, we use these extended UML class diagrams to illustrate information structures that emerged from the application of Hybrid Wikis in different enterprises.

3.4.2. Custom views

Built-in views (cf., Section 3.4.1) in Hybrid Wikis give users immediate access to structured information with little effort. However, in some cases these views are not sufficient since they most likely contain information which is not relevant for the specific demands of an individual business user. For instance, the type table view is restricted to exactly one type and always contains all attributes ordered by the frequency of the values. But a user might be interested in a tabular view related to more than one type only showing of a few attributes in a specific order. This is achieved by providing user-specific views, that is, views that can be customized according to a user’s specific information needs.

All custom views are based on structured queries as introduced in Sections 3.4.3 and 3.2.4.1. This means different search filters can be combined in order to obtain a set of relevant content objects. Based on this set, the view additionally filters (by using projection) irrelevant structured information according to the user’s needs. That is, a view basically consists of two filter parts, a search filter (or multiple) and a projection filter. Hybrid Wikis support two kinds of custom views.

The list view produces a simple list according to the underlying query’s search result showing a link for each hit (cf., Figure 3.36 (1)). No structured information is shown in the list, that is, the list view can be considered as a projection that filters all structured elements.

The custom table view produces a table according to the underlying query’s search result showing a link for each hit in the first column (cf., Figure 3.36 (2)). Additionally, users can customize which attributes are shown as table columns in which order (e.g., Budget and End

3. Hybrid Wikis

The screenshot shows a wiki page for 'Hans Mayer'. At the top, there are navigation links: Wikis » Project Example Diss » Hans Mayer. On the right, it says 'Last editor: Christian Neubert - 1 minute ago'. Below the navigation are tabs: View, Details, Attachments, Versions. On the right side, there are links: Edit, Watch, Browse this Wiki, Delete, New Page, Clone. The main content area is titled 'Hans Mayer' and includes tags: manager, profile. Under 'Background', it says 'Hans Mayer is project manager since 2010. He is particularly interested in discussions about risk management.' Under 'Managed Projects', there is a search bar and a table. The table has columns: Budget (2), End (2). The first row is 'Calculation' with a budget of 300.000 and end date 06.05.2012. The second row is 'Save the whales' with a budget of 20.000 and end date 06.05.2011. Below the table, there is a '+ (5)' button. Under 'Active Projects', there is a list: Project Documentation (1), Development Project Diss. On the right side, there is a 'Types' section with 'staff' and 'internal'. Below that is an 'Email' field with 'hans.mayer@hybrid-wiki.com'. Then, there are fields for 'Firstname' (Hans), 'Lastname' (Mayer), 'Personnel Number' (1.004), and 'Position' (Junior Project Manager). There are also fields for 'Company' and 'Decision Maker'. Below that is an 'Incoming Links' section with 'Manager of (2)' (Calculation, Save the whales) and 'Member of (3)' (Development Project Diss).

Figure 3.36.: Custom table and list view embedded in the built-in content of a wiki page.

as in Figure 3.36). They can also define which column to use for sorting the search results (cf., Figure 3.36 (3)). It is important to note that a hit not necessarily has to provide one of the attributes specified by the view's projection filter. For instance, if a view filters by attribute *budget* and the search result does not contain any content object having an attribute *budget* then *budget*'s table column only consists of empty cells, which are presented as input fields. This way, users are encouraged to fill out these empty fields. Even if none of these objects provide an attribute *budget* as in this case, filling out an empty field results in the creation of an attribute with key *budget* within the content object underlying the table row. Therefore, the custom table view significantly contributes to managing and providing structured information consistently. Finally, users can add new table entries (e.g., structured wiki pages) according to the query underlying the table (cf., Figure 3.36 (5)). For instance, the custom table view depicted in Figure 3.36 filters for finished projects managed by Hans Maier. Then adding a row means that a new content object is created having two attributes, *status* with string value *finished* and *manager* with a link to *Hans Maier*. Subsequently, the new object is shown in this table providing empty input fields for the attributes *Budget* and *End*.

A third kinds of views based on structured information are graphical. However, since graphical views are not provided by Hybrid Wikis built-in¹⁵, they are introduced later in this thesis (cf., Section 6.1.1).

All search-based custom views (i.e., the query and view configuration) can be embedded in the full text of any content object. When displaying such an object the underlying query is executed and the search results are depicted for example as a custom table (or a simple list) according to the view's filters. By combining multiple such views it is possible to configure

¹⁵Built-in here refers to the basic capabilities of Hybrid Wikis. From a technical perspective, graphical views are only available by using a specific plugin (cf., Section 4.2.2).

a content object that can be used as a dashboard [Ma05] with little effort. This custom dashboard empowers users to observe relevant and business critical information according to their specific needs. For instance, the dashboard can be utilized to monitor violations of integrity constraints.

3.4.3. Structured search

A general search interface allows a faceted drill-down [Ha10, Yi08] based on the structured contents (cf., Figure 3.37). By means of this interface, users can specify queries by combining different type- (cf., Figure 3.37 (1)) and attribute-filter (cf., Figure 3.37 (2)). Furthermore, the interface allows to search for invalid values, that is, values violating integrity constraints (cf., Figure 3.37 (3)). A query configuration can then be embedded in the wiki text (cf., Figure 3.37 (4)).

The screenshot shows a search interface with a breadcrumb trail: [Home](#) » [Search](#) » [Space: Project Example Diss](#) » [Type: project](#) » [Status: completed](#) » [Manager: Hans Mayer](#). A link "Embed in wiki page" is visible in the top right.

On the left, there is a faceted drill-down menu with the following sections:

- Content type**: Wiki Page (2)
- Space**: Project Example Diss (2)
- Type**: project (2) (1)
- Status**: completed (2) (2)
- Manager**: Hans Mayer (2)
- Additional Attribute**:
 - Budget (2)
 - Contact (1)
 - End (2)
 - Members (1)
 - Start (2)
 - Sub-project (1)
- Special**:
 - Contains Invalid Links
 - Contains Invalid Values (3)

The main content area displays the search results for the query "research project todo". It shows "Results 1 - 2 of 2" and a "Search" button. The results are sorted by "Relevance".

The first result is titled "Calculation" and includes a "Text..." field. The second result is titled "Save the whales" and also includes a "Text..." field. Both results show the page title "Project Example Diss" and the editor's name "Christian Neubert" with the time since last edit.

At the bottom right, it indicates "1 (total result pages: 1)".

Figure 3.37.: The search interface allows a faceted drill-down based on the structured contents.

The configuration resulting from the example depicted in Figure 3.37 is shown in Figure 3.38. The query (i.e., search filter) is given in the first part of this configuration (cf., Figure 3.38 (1)), the second part specifies how the search results are presented (cf., Figure 3.38 (2)). In this example, the results would be displayed as a table with two columns (*Budget*, *End*) in

3. Hybrid Wikis

descending order by *Budget*. This configuration is also underlying the custom table view depicted in Figure 3.36.

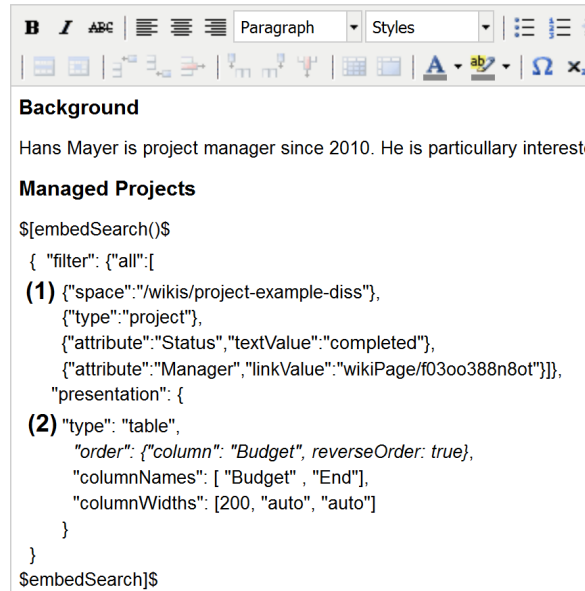


Figure 3.38.: Query and view configuration embedded in wiki text.

3.4.4. Advanced UI operations

In [Po07] in-place editing is described as a technique that gives “web page readers with editing permissions the ability to ‘live edit’ a page’s modifiable fields without having to go to a separate web page or form”. Hybrid Wikis support in-place editing for all structuring concepts, that is, attributes, type assignments, types, attribute definitions, and constraints. The main purpose in supporting in-place editing is to encourage users to provide new information structures and manage existing ones in a lightweight way.

On the instance level, types (i.e., type assignments) as well as attributes with keys and values can be added, changed, and deleted quickly and easily by means of in-place editing (i.e., with a few clicks in the UI). Furthermore, attribute suggestions can also be filled out ‘in-place’ (cf., Figure 3.27 (10)). Since it is convenient to contribute new attributes and change types using in-place editing techniques for individual content objects the bottom-up evolution of an information schema is additionally facilitated.

It is important to note that in-place editing of the content objects’ structures is orthogonally applicable to all built-in views (cf., Section 3.4.2) and custom views as well (cf., Section 3.4.1). For instance, it is possible to change (add, delete) an attribute’s value ‘in-place’ even if it is shown in a built-in type table view (cf., Figure 3.34) or as part of a custom embedded table view within a dashboard (cf., Figure 3.36 (4)).

In order to ease the transitions introduced in Section 3.3.2 we propose using the technique of drag and drop [LSA11]. Since structured and unstructured information in all views is clearly separated from each other transitions between them are well suitable to be supported

by drag and drop, for example transitions to create new attributes from unstructured text (cf., Section 3.3.2.2) or creating a type from an attribute (cf., Section 3.3.2.3). However, in Hybrid Wikis drag and drop is currently only supported for the transitions between

- tags and types (i.e., transforming a tag into a type),
- unstructured content (including links) and attributes, and
- two attributes (i.e., merging attributes).

The other transitions are supported by conventional UI elements (e.g., combo boxes) or have to be performed manually.

Implementing Hybrid Wikis in Tricia

In this chapter, we present the current implementation of Hybrid Wikis. In particular, we describe how the concepts and techniques introduced in Section 3 are realized based on the Enterprise 2.0 platform Tricia. In Section 4.1, we explain the underlying system architecture and the interfaces to the base platform. In Section 4.2, we introduce the extended architecture and discuss important design decisions from a technical point of view.

4.1. Introduction to Tricia

In the following, we give an overview of Tricia, a framework for the development of web applications. We introduce the core concepts of the framework relevant for the implementation of Hybrid Wikis as described in Section 4.2. First, in Section 4.1.1 we give an overview of Tricia's architecture by explaining the framework's constituents supporting the model-driven development of web applications¹. In particular, in Section 4.1.2 we focus on the data modeling framework which enables the modeling of the application's domain objects, such as wiki pages or blog posts. Subsequently, in Section 4.1.5 we sketch how existing applications can be customized by using plugins and extension points. In Section 4.1.6 we briefly introduce an Enterprise 2.0 application built based on the Tricia framework which is commercialized by the InfoAsset AG². This application is also named Tricia, since it is currently the only productive implementation based on the web application framework Tricia. This implementation is the foundation for the development of Hybrid Wikis.

¹Tricia's frameworks as introduced here are originally described in [BMN10a].

²<http://www.infoasset.de>; visited on January 13th 2012.

4.1.1. Architecture

Figure 4.1 provides an architectural overview of a typical web application implemented on the Tricia platform using a notation similar to a UML deployment diagram. Such an application provides *Hypertext Transfer Protocol* (HTTP)(S) access for its web clients possibly including *Asynchronous JavaScript and XML* (AJAX)-style asynchronous interactions, a *Representational state transfer* (REST)-ful web *Application programming interface* (API) to allow third parties to query and update the content managed by the application, and a model introspection [BM06] interface to allow third parties to discover and query the data model, the access control model, and the interaction model implemented by the application. Tricia's current implementation supports a single server (up to fifteen page requests per second on stock hardware [Ad09]) but the architecture is designed for a scale-out to multiple servers using a cluster database. A Tricia application requires a Java 1.6 runtime environment on Windows or Linux, a database server, and a Lucene full text search engine [MHG10]. Currently Tricia supports MySQL [WAM02], Oracle [PM07], and for testing purposes the in-memory database HSQLDB³. There also exists a prototypical implementation which persists data using the NoSQL database MongoDB [DC10].

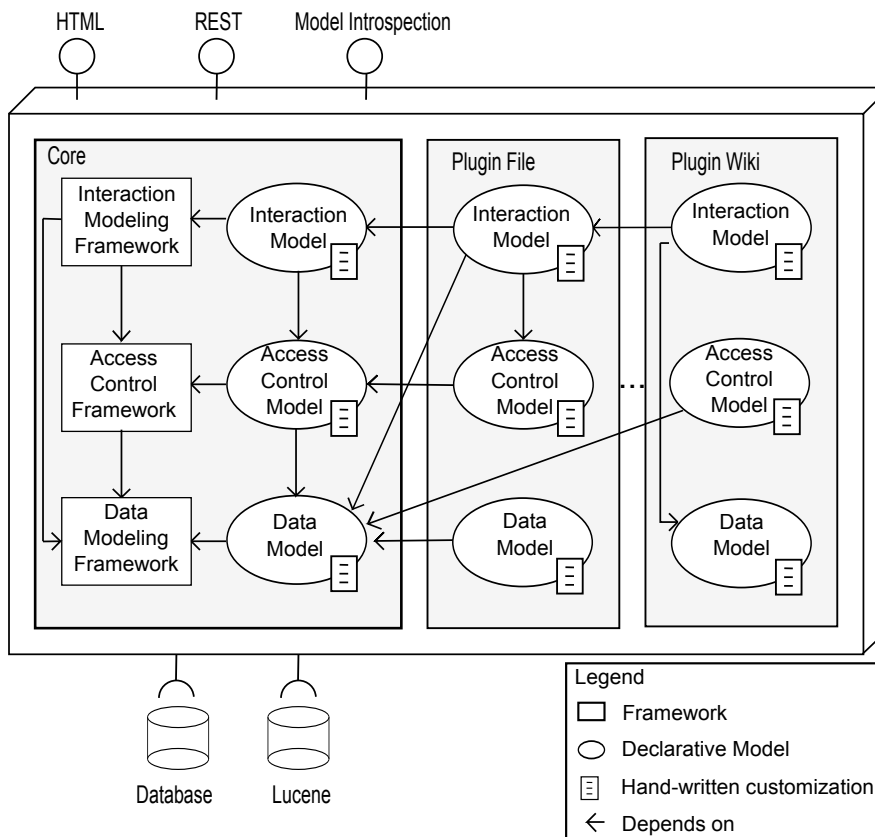


Figure 4.1.: Architectural overview of a typical web application implemented based on the Tricia platform.

A Tricia application consists of a core and one or more plugins that define the application

³<http://hsqldb.org>; visited on January 13th 2012.

in a modular fashion (cf., shaded areas in Figure 4.1). Each plugin specifies the plugins it depends on. Cyclic dependencies between plugins are not allowed and are detected at construction time. For example, the wiki plugin depends on the file plugin, since files may be attached to wiki pages and wiki management thus requires file management. The core defines abstractions required by virtually all applications of the domain, for example user profiles, groups, memberships, login and registration procedures. Both plugins, wiki and file, use such user profiles to identify the last editor of a wiki or a file. Other abstractions provided by the core are discussed in Section 4.1.2.

Each plugin and the core define a data model, an access control model, and an interaction model. Each model defines a fragment of the data structures and behavior of the entire application. These models are expressed by graphical and textual notations and are available at runtime for introspection (cf., ovals in Figure 4.1). If necessary, they can be augmented by customizations written in Java (e.g., to express business logic). Models from a plugin p may reference models in p and in plugins imported by p (cf., arrows in Figure 4.1). The following rules apply: Interaction models may reference other interaction models, access control models, or data models. Access control models may reference other access control models or data models. Data models may reference other data models only.

The core is provided as part of the Tricia platform and consists of three layered Java frameworks (cf., layered architecture in [Bu96]) for data modeling, access control, and user interaction (i.e., views and controllers). Each framework provides abstractions and extension points, which have to be instantiated or customized in order to build a Tricia application. Frameworks are developed and maintained by the Tricia core developers as part of the core development process, customizations are developed by application developers as part of the application development process [Fr98]. There are two different kinds of customizations. The majority of customizations can be done in a declarative, model driven way. This results in models to be created. For some aspects to be customized it is more convenient to specify them using the full expressive power of the base language, which is Java in our case. An example for this kind of customization is complex business logic. Figure 4.1 emphasizes the central role of the data modeling framework as the foundation for model driven web application development. In the following sections, we focus on the concepts of the data modeling framework, since most of the extensions provided by Hybrid Wikis are related to the data model.

The following examples should suffice to highlight the use of the application-specific data model in all frameworks:

- For each entity type, the Tricia interaction framework can generate multilingual element-oriented CRUD views (i.e., create, read, update, delete) and set-oriented table controls. These views may include rich text and media attachments (e.g., images, files).
- Associations between entities can be navigated in an element-oriented (via hyperlinks) or set-oriented (declarative queries) style. End users can interactively create full text and structured queries for entities of a given type or any type.
- Tricia can also expose these views and controllers as REST-ful web APIs to allow external systems to interact with Tricia applications.
- The Tricia access control framework allows application developers or end users to associate access control policies with entity types or even individual entities. These policies

4. Implementing Hybrid Wikis in Tricia

can restrict read, write, and administration rights to user groups or to individual users (role based access control or discretionary access control). The policies are automatically enforced at the user interface and at the web API level.

- The Tricia data modeling framework automates the data migration steps necessary after (series of) typical incremental schema changes.

4.1.2. Data modeling framework

In the following, we incrementally introduce the elements and concepts of the Tricia data modeling framework as shown in Figure 4.3 by means of a simplified model taken from the domain of social software web applications. In [BMN10a] the data modeling capabilities of Tricia are described in more detail.

The example data model is presented in Figure 4.2. It consists of the concepts wiki and wiki page. The application based on this model allows registered users to manage a collection of wikis. Each individual wiki contains an arbitrary set of wiki pages. Wikis and wiki pages both are identified by a unique name. For instance, this name is shown in the web user interface when listing all available wikis or all wiki pages of an individual wiki respectively. Furthermore, a readable, structured, and persistent URL is provided by both, wikis and wiki pages. One of the wiki's pages can be defined as the home page. The home page represents the default view of the wiki and is shown when the URL of the wiki is called. The content of a wiki page consists of rich (hyper-)text, that is, markup, embedded hyperlinks, and attached media files (cf., Section 3.1.2). The given sample application consists of an individual plugin with a data model defining the entities wiki and wiki page. Subsequently, we describe how developers can build this application, in particular the application's data model, by means of the concepts and elements provided by the Tricia data modeling framework.

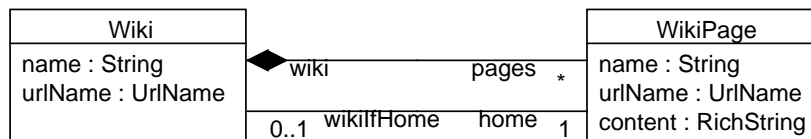


Figure 4.2.: Example data model consisting of wikis and wiki pages and their relations.

Tricia's domain objects are represented as objects of type *Entity*. Entities provide a name and an internationalized label. In our example application two entities are given, wiki and wiki page. The name of an entity identifies the concept in the data model, in our example the concepts' names are wiki and wiki page. The internationalized label is used to generate custom views according to the users' language.

Properties of domain objects are represented as objects of type *Property*. The data modeling framework provides the following predefined basic property types: *BooleanProperty*, *IntProperty*, *StringProperty*, *DomainValueProperty*, *DateProperty*, *TimestampProperty*. Each property type may introduce certain attributes, which can be customized. For instance, *StringProperty* represents a character sequence with a size limited by the *maxLength* attribute. The

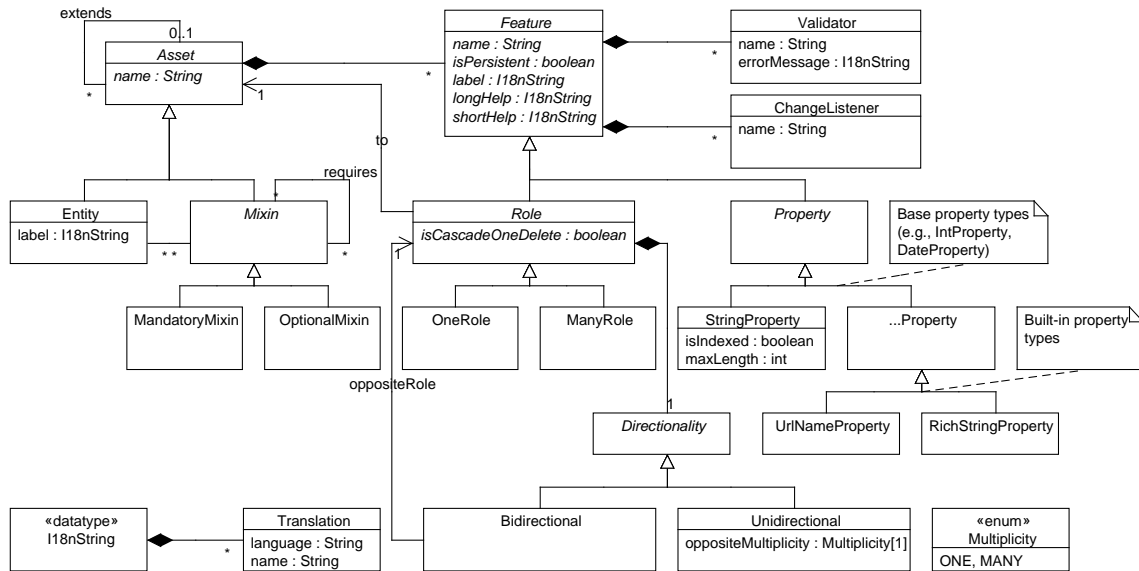


Figure 4.3.: Concepts of Tricia's data modeling framework.

attribute *isIndexed* indicates whether an index should be created to speed up value-based queries for that property. Building on the basic property types (e.g., *StringProperty*) the Tricia core provides the following domain-specific property types (extract):

- A *RichStringProperty* is a subtype of *StringProperty*, which holds HTML content (cf., Section 3.1.2). The implementation of *RichStringProperty* ensures that the content does not contain malicious scripts, automatically detects dangling hyperlinks, and supports a consistent application-wide URL renaming.
- A *UrlNameProperty* is used to provide meaningful URLs for domain objects (cf., Section 3.1.2.3). URLs should match as closely as possible the name of the object, but may be subject to additional constraints due to character set limitations for URLs.

In our example (cf., Figure 4.2), the properties of a wiki are *name* of type *StringProperty* and *urlName* of type *UrlNameProperty*. A wiki page also has the properties *name*, *urlName*, and additionally a *content* property of type *RichStringProperty*.

Associations between domain objects are represented in Tricia by modeling the association ends as objects of type *Role*. A role specifies the type of the associated entity, which is represented in the data model framework of 4.3 by the *to* reference. There are two kinds of multiplicities: A single-valued association is modeled using the class *OneRole* and a multi-valued association through the class *ManyRole*. The directionality of a role is expressed by the mandatory concept *Directionality*. Bidirectional roles reference the corresponding opposite role. In this case, the multiplicity of the counterpart is implicitly given through the type of the opposite role instance (i.e., *OneRole* or *ManyRole*). Since a unidirectional role does not specify an opposite role, its multiplicity cannot be derived and has to be explicitly defined through the *otherMultiplicity* attribute. The attribute *isCascadeOnDelete* indicates to delete

the referenced entities if the owning entity is deleted (cf., composition in UML). In our example (cf., Figure 4.2), a wiki is used as a container for a set of wiki pages.

Properties and roles share some common attributes, which are captured by the abstract super concept *Feature*. Each feature has a name, an internationalized label (cf., entity attributes), as well as the two internationalized attributes *longHelp* and *shortHelp*. These labels are used in generated views to describe the meaning of a feature to end users in their own language. The flag *isPersistent* indicates whether the value of a feature is persistently stored in a database, by default this flag is set. A non-persistent property can for example be used for derived values, which are calculated depending on the values of other persistent properties and can be shown in certain views. The Tricia data modeling framework also supports inheritance, that is, a derived entity inherits all features of its parent entity. By default, a single-table strategy [Fo03] is used to map the inheritance tree to a single database table.

An important aspect of data modeling is the specification of constraints to ensure data integrity. In the Tricia data modeling framework constraints can be modeled through *Validators*. As part of the declarative model, a validator has a name and provides error messages, which are shown to end users in case of a validation failure. Validators can be specified for all features, that is, for both, roles and properties. As an example, a validator verifies whether the value of a *StringProperty* satisfies a specific pattern (e.g., email address). An example for role validation is to constrain the cardinality of an association. In our example, a wiki page has to be part of a wiki. This can be realized by a role validator applied to role *wiki*. The Tricia data modeling framework provides a set of built-in property validators (e.g., *EmailValidator*, *MinimalLengthValidator*) as well as predefined role validators (e.g., *NotNullOneValidator*, *NotEmptyManyValidator*). Validators can be parameterized with values.

In the Tricia data modeling framework *ChangeListeners* are used to propagate data model changes through the system. A change listener has a name and is registered on a feature in order to be notified when the value of the feature changes. Change listeners apply to both kinds of features, that is, for both, roles and properties. For example, a change listener *updateUrlName* can be defined for the name property (i.e., *StringProperty*) of a wiki page. Then, if the name property is set (i.e., the name changes) for a newly created page and no URL is given by the end user, the value of the name property is used as the URL's default. This operation is performed by the change listener, which is notified when the name property changes.

The only way of realizing reuse at the data model level introduced so far is the mechanism of inheritance. Since models in Tricia are realized by subclassing framework classes, this mechanism is constrained by having a single inheritance chain, which means that an entity can have only one entity it inherits from. This imposes a severe limitation, and is not sufficient for real-world modeling problems. To enable reuse on a more fine-grained level, Tricia utilizes the concept of *Mixins* [ALZ03]. Mixins extend entities with additional properties and roles. As shown in Figure 4.3, the entity and mixin classes are subtypes of the abstract class *asset*, which provides the capability of having features. Mixins can be assigned to other entities and vice versa, which is expressed by a many-to-many association between entity and mixin as shown in the class diagram in Figure 4.3. We distinguish two kinds of mixins, which are realized by the framework classes, *MandatoryMixin* and *OptionalMixin*. Mandatory mixins are statically assigned to a certain entity and cannot be removed at runtime.

Table 4.1.: Mandatory mixins and their usage in the core plugin and in the wiki plugin.

	Linkable	Searchable	Taggable	Commentable	Versionable
Group	x	x	x		
Membership					
Person	x	x	x		
Principal	x	x			
Comment	x	x			
Search	x	x	x		
Version	x				
Wiki	x	x			
WikiPage	x	x	x	x	x

In Table 4.1 an extract of existing mandatory mixin types and their use by entity classes is shown. These mixins enable fine-grained reuse of cross-cutting aspects. A mixin can depend on other mixins, for example a searchable entity (i.e., an entity using the mandatory mixin *Searchable*) requires to be linkable (i.e., to have a URL) too, otherwise the asset cannot be accessed if it is shown for example in a search result list. In this example, it is not permitted to define searchable entities, which are not linkable.

In contrast, optional mixins can be assigned to objects and can be removed at runtime by end users. An example of an optional mixin is the class *CalendarItem*, which can be assigned to wiki pages. It marks the assigned wiki page as representing a temporal event, which is characterized by additional features such as *startDate*, *endDate*, and *eventCategory*. As opposed to mandatory mixins, this capability is not required for all wiki pages, but can be assigned by end users at runtime. The existence of this mixin type then indicates whether a specific wiki page is displayed in a calendar view. As shown in Table 4.1, the Tricia core includes predefined entity types which are essential for the domain of enterprise web applications. They comprise entities for modeling users and user groups: *Person*, *Group*, *Membership*, and *Principal*. These entities are the foundation for the access control framework (cf., Section 4.1.4). Other built-in entities are *Link*, *Comment*, and *Version*, which are associated with the respective mandatory mixin types. For example, the mixin *Commentable* establishes an one-to-many association to entities of type *Comment*.

Tricia's data modeling framework as introduced in Section 4.1.2 represents an abstraction layer in order to develop the domain objects of the web application. This layer also ensures an encapsulation of the domain objects from the physical access to the data storage. The data modeling framework provides a built-in mechanism to map an entity with its properties, mixins, and roles to corresponding elements in the data store. Currently, Tricia's default storage implementation is an object relational mapping of the entities and its constituents to corresponding *Structured Query Language* (SQL) elements, in particular to MySQL elements. Entities are mapped to database tables, properties to table columns, and roles to foreign key constraints or tables in case of m:n-relations. Since a mixin does not have an own object identity its properties and roles are stored as part of the table of its owning entity. For instance, if a wiki page entity supports comments by means of a *Commentable* mixin and this mixin provides a *StringProperty* to represent (i.e., store) the comments, this property will be

mapped to a column of the wiki page's database table. Additional storage implementations supporting object relational mapping in SQL are Oracle, MSSQL, DB2, HSQL.

Besides object relational mapping, the Tricia framework provides a prototypical implementation of the open source database MongoDB, a key-value store supporting document-oriented database access. The class of document-oriented databases in literature is often referred to as NoSQL databases or structured storages [Ch08a, LM10].

4.1.3. Interaction framework

The Tricia framework is based on the architectural pattern *Model-View-Controller* (MVC) [BD09]. The model in this pattern is represented by a domain object (i.e., an entity, such as wiki page or blog post) from the modeling framework (cf., Section 4.1.2). The view and the controller are represented by concepts provided by the interaction framework (cf., Figure 4.4). A Tricia handler⁴ is responsible for the processing of an individual client's request. A handler primarily contains the business logic. Additionally, a handler checks if the requesting client is authorized to execute this business logic (cf., Section 4.1.4). A template⁵ is in most cases a HTML page which is returned to the client according to its request.

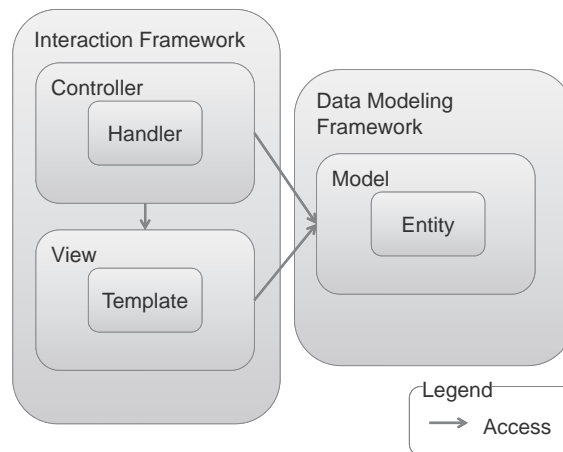


Figure 4.4.: MVC pattern realized in Tricia.

4.1.4. Access control framework

Access control in Tricia basically works as introduced in Section 3.1.2.2. This means it can be specified who is allowed to read and to write content objects by using a set of principals as readers and writers. In particular, individual entities can be protected from unauthorized read access by using the mixin *ReadProtected*⁶.

⁴<http://www.infoasset.de/wikis/javadoc-import-wiki/platform-handler-handler>; visited on January 13th 2012.

⁵<http://www.infoasset.de/wikis/javadoc-import-wiki/platform-documentation-templatedoc>; visited on January 13th 2012.

⁶<http://www.infoasset.de/wikis/javadoc-import-wiki/platform-assets-readprotected>; visited on January 13th 2012.

4.1.5. Plugins and extension points

Plugins in Tricia can be used in order to provide functionality in a modular fashion. A plugin can contribute new concepts, such as entities, templates, and handler. In addition, it is also possible to extend existing concepts from other plugins to customize them according to specific demands. This is achieved by using extension points provided by the Tricia framework⁷. For instance, an extension point within a plugin depending on the wiki plugin (cf., Section 4.1.2) can specify an additional property for a wiki page. Since the entity's additional properties are likely to be visible in a view it is also possible to extend templates and handler.

4.1.6. Enterprise 2.0 platform Tricia

The Enterprise 2.0 platform Tricia is the basis for the development of Hybrid Wikis. Figure 4.5 depicts the plugins provided by the Tricia platform before the development of Hybrid Wikis. The core plugin contains predefined entity types, such as persons and groups (cf., Section 4.1.2). Based on the core, the file plugin supports the management of documents and folders. The wiki plugin mainly consists of the entities wiki page and wiki. A wiki represents the container for a set of individual pages. The relation between blog and blog post is the same as between wiki and wiki pages. Documents, wiki pages and blog posts represent the content objects of Tricia, wiki, blog, and folder their corresponding spaces (cf., Section 3.1.1). Since principals (i.e., persons and groups) are globally available in the application (cf., Section 3.1.2.2) they are part of a special system space. However, they can also be considered as content objects, since both provide a full text description.



Figure 4.5.: Plugins provided by Tricia before Hybrid Wikis.

Tricia provides a set of built-in services orthogonal applicable to the content objects provided (i.e., wiki pages, blog posts, and documents). Services are for example tagging, versioning, and commenting. The complete service catalog provided by the Enterprise 2.0 platform Tricia is described in [BMN09]. How the services relevant for this thesis can be applied to attributes, types, attribute definitions, and constraints is discussed in Section 3.2.4.

4.2. Implementing Hybrid Wikis

In this section, we explain how Hybrid Wikis are realized based on the Enterprise 2.0 platform Tricia. We introduce the data model the implementation is based on in Section 4.2.1 and

⁷<http://www.infoasset.de/wikis/javadoc-import-wiki/platform-services-extension>; visited on January 13th 2012.

Tricia’s extended architecture in Section 4.2.2. Subsequently, we technically explain how suggestions can be calculated efficiently (cf., Section 4.2.3) and illustrate how transitions (cf., Section 4.2.3) and consolidation techniques (cf., Section 4.2.5) are supported in Hybrid Wikis. In Section 4.2.6 we describe how information structures can be exchanged by means of import and export interfaces. In Section 4.2.7 we show that the concepts of Hybrid Wikis support federated data integration scenarios. In particular, we demonstrate how information objects from third-party business applications can be accessed and extended by means of Hybrid Wikis. This way, we show how different information islands within enterprises can be integrated.

4.2.1. Data model

Figure 4.6 depicts the data model that the current implementation of Hybrid Wikis is based on. The data model is quite similar to the conceptual model introduced in Section 3.1. In the following, we describe the main differences between both models and discuss important design decisions from a technical point of view.

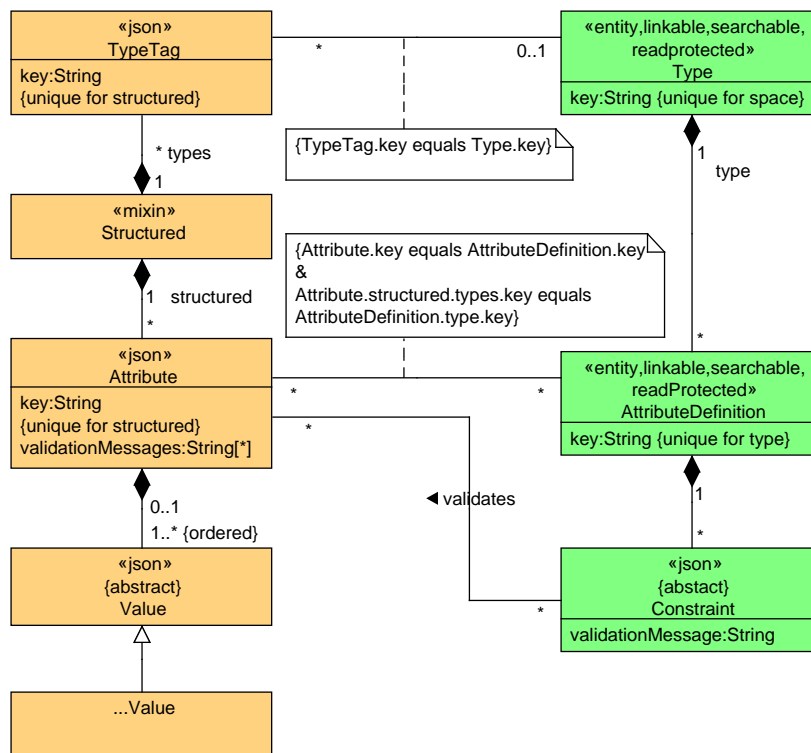


Figure 4.6.: Data model underlying Hybrid Wikis.

4.2.1.1. Using mixins for structuring support

In order to support information structuring we decided to implement a mixin (cf., Section 4.1.2). By doing so, information structuring is applicable to any kind of content object and therefore can be considered as an orthogonal Enterprise 2.0 service, similar to other ser-

vices such as tagging or commenting [BMN09]. For instance, the mixin can be applied to wiki pages. However, not every content object needs to use this mixin (cf., Figure 4.7). The mixin *Structured* is depicted in Figure 4.6. It primarily encapsulates the ability to assign types and attributes to content objects. Therefore, it can be considered as the concrete implementation of the concepts introduced in Section 3.1.6. But using a mixin in consequence also means that concepts supporting information structuring always have to be of type entity, since a mixin can only be applied to entities according to the Tricia data modeling framework (cf., Section 4.1.2). In particular, this design decision has an impact on the realization of structured links since attributes and values are not stored as first class entities. However, this is discussed in Section 4.2.1.5 in detail.

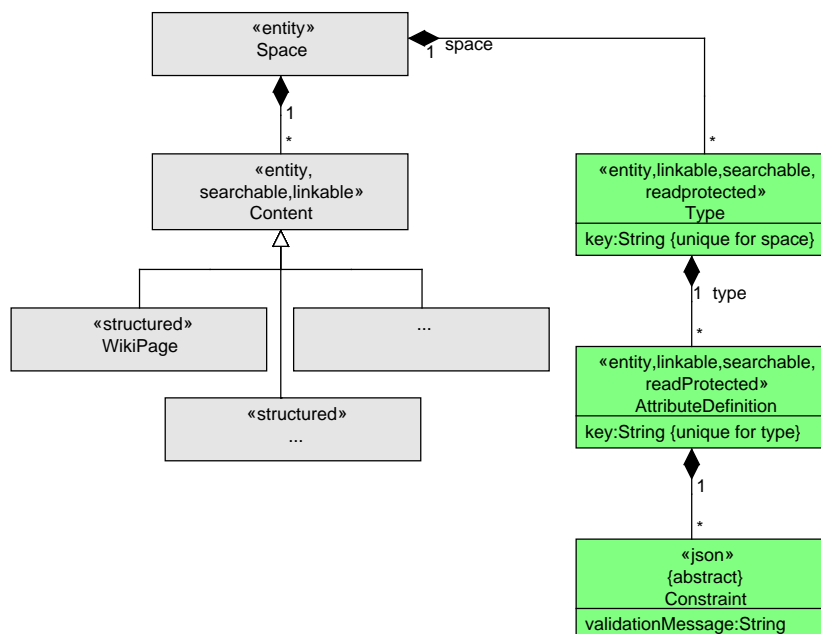


Figure 4.7.: Mixin *Structured* applied to some content objects.

4.2.1.2. Type tags

The main difference between the data model shown in Figure 4.6 and the conceptual model introduced in Section 3.1 is that two concepts for the representation of types exist in the implementation. The type represents the concept according to Section 3.1.4 and the type tag an assignment of a type. A type tag is similar to a plain tag (cf., Section 3.2.4.2) with the difference that it serves as a classifier. Assigning it means to explicitly make a statement about a content object's type by using a tag.

The management of type tags is completely handled by the tagging module provided by the Tricia platform, including

- UI controls,

4. Implementing Hybrid Wikis in Tricia

- writing type tags to the datastore, and
- indexing type-tagged content by means of the Lucene search engine.

Only little extensions were applied to the tagging module. In order to distinguish between plain tags and type tags a namespace is used when assigning or querying tags (cf., Figure 4.9). When assigning a type tag to a content object a different namespace is specified as if assigning a conventional tag. That is, by means of the namespace type tags and normal tags are distinguished. Figure 4.8 exemplifies the assignment of a type by means of tagging services. Since the current implementation of Hybrid Wikis is based on tagging services the mixin *Structured* depends on the mixin *Taggable* as indicated in Figure 4.9. Additionally, the plugin providing structuring support (cf., Section 4.2.2) contains some style sheets (e.g., in order to represent type tags in a different color within the tagging UI controls).

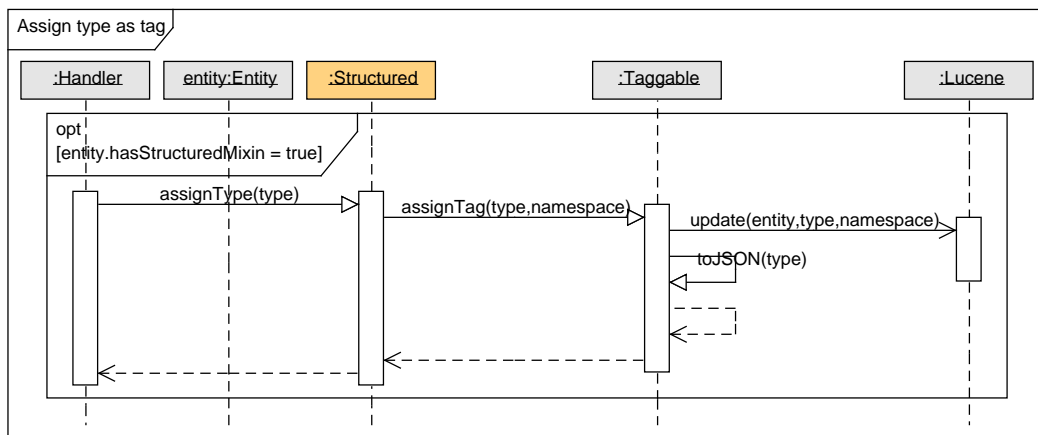


Figure 4.8.: Assignment of a type by means of tagging services.

We decided to support typing of content objects by using tagging services due to the following reasons: As explained in [GH05], what-it-is tags are used for the classification of resources. However, when applying conventional tags only it is not possible to explicitly make a statement about the content object's type. This means type tags contribute to make implicitly given what-it-is tags explicitly visible. Additionally, users immediately gain benefits from type tags by means of built-in views, better suggestions, and the ability to specify more precise queries. Furthermore, users of an Enterprise 2.0 platform are likely to be familiar with tagging services. Therefore, they are also familiar with the existing tagging UI control and only need to learn what the benefits are when using type tags instead of conventional tags in Hybrid Wikis. From a user's perspective, type tags are lightweight means to classify a resource on the instance level. This is due to the fact that they are simple keywords (i.e., similar to conventional tags) and users do not necessarily need to act as a tailor when assigning them. That means neither users need to create a type definition (with description, attribute definitions, and constraints) nor they have to know which types already exist⁸ before assigning a type tag. Our assumption is that editors are more familiar with tagging an object than with schema management activities and therefore less inhibited in assigning a type tag. Type tags facilitate

⁸Even if users not necessarily need to know the already existing types they are supported by autocompletion to choose one of them.

the classification (i.e., typing) of a resources for authors, tailors can subsequently specify that classifier (i.e., type) more precisely (e.g., by providing a description, specifying further constraints, or binding attributes to the type).

As depicted in Figure 4.8, when using the tagging module type assignments are automatically serialized to the database (as *JavaScript Object Notation JSON* (JSON)) and additionally indexed by means of the Lucene search engine. An important reason for using tag-based types is the performance of the Lucene index when querying data. As discussed in [JZW09], the Lucene engine in most cases is faster than a relational database when performing exact queries, even when using indexes in the database. Exact here means that a given value equals the value of a field specified in a query. Exact queries in this thesis are required to determine the set of content objects with a specific type, for example in order to generate a type table view showing all instance of a selected type (cf., Section 3.4.1). For instance, in order to determine all projects in the application it is required to define a query selecting all content objects exactly typed as *project*.

A conceivable alternative to using tags in order to type a content object would be to define a Tricia many role (m..n) between the mixin *Structured* and the entity *Type* (cf., Figure 4.6). However, in this case it would be necessary to join the table underlying the mixin's owning entity with the table of *Type* in order to determine a set of specifically typed objects. Additionally, more than one content object can be structured. This means a set of specifically typed objects can only be determined by executing multiple join operations when querying against a relational database. Since performing multiple joins in a relational database is certainly more expensive than executing a query in a flat key-value store asking for a specific field (representing the type tag) we decided to (re)use the Lucene-based querying capabilities of the tagging module in order to determine a set of specifically typed content objects. Furthermore, the management (e.g., persistence management) of simple keywords (i.e., type tags) is more lightweight than the management of first class entities. For instance, in case of type tags removing a type from a content object (i.e., changing an object's type assignments) means that only the content object itself needs to be updated in the database. This is due to the fact that the serialized type tags are stored as part (i.e., as a property) of that object. In case of using first class entities, it has to be determined if the removed type is still related to other objects (by using database queries). Only if not, the type can be deleted. Therefore, from a technical point of view, when using entities changing an object's type assignments is more heavyweight than in case of type tags.

The tagging module additionally allows to specify queries selecting objects with multiple tags. This mechanism can also be used to query content objects having multiple types. However, it is currently only possible to specify conjunctive queries. For instance, it is not possible to ask for content objects typed as *project* or typed as *research project*. This is due to the fact that the tagging module is limited to conjunctive queries although the Lucene index also supports disjunction. Another advantage of using Lucene for defining queries is its scalability⁹. This is especially important in case of growing user-generated (structured) content.

The tagging module provides tag suggestions built-in. Therefore, types are automatically suggested when reusing the existing tagging mechanisms. It is only necessary to restrict the namespace to type tags when calculating suggestions. However, the built-in mechanisms only

⁹<http://www.lucidimagination.com/content/scaling-lucene-and-solr>; visited on January 15th 2012.

generate type suggestions based on multiple types and based on normal tags. The other type suggestions as proposed in Section 3.3.1.3 are not implemented up to now.

As indicated in Figure 4.6, type tags are not shared across multiple content objects since they are literals. That means, every content object provides its own set of type tags (i.e., its type assignments). Additionally, type tags and types are not connected by using a Tricia role (cf., Section 4.1.2) since tags are not stored as first class entities in Tricia. Therefore, the determination of types related to a content object requires to query the datastore. For instance, it is required to execute a database query in order to show a user the description of a type as help text when assigning a type tag. In this case, only one additional database is required to determine the current object's schema information (e.g., description from a type or related attribute definitions). Therefore, this operation is not critical for a single object. However, in case of more than one object executing multiple database queries can cause performance issues. For instance, validating the values shown in the type table view (cf., Section 3.4) would require to execute a database query for each cell¹⁰ in order to determine the constraints related to the attribute underlying the cell. In Section 4.2.1.6, this potential performance issue is addressed by caching schema-based information directly in the object.

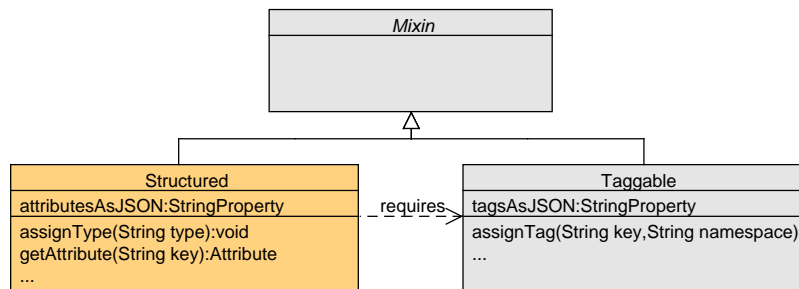


Figure 4.9.: The mixin *Structured* is based on the mixin *Taggable*.

Another difference between the data model (cf., Figure 4.6) and the conceptual model (cf., Section 3.1) is that it is not possible to define an order for type assignments. This is due to the fact that Tricia's tagging module does not support specifying an order for tags, by default they are ordered alphabetically.

4.2.1.3. Attribute order

As indicated in Figure 4.6, a content object's attributes and a type's attribute definitions are not ordered. As discussed in Chapter 3, the definition of an order for attributes requires

- specifying an order for individual content objects,
- passing the order of typed content objects to the type, and
- consolidating diverging orders by applying the type's order to its instances.

¹⁰This can be optimized by caching the attribute definitions for each table column.

These harmonization techniques are required since content objects can be related to multiple types with different or even conflicting orders. For instance, a content object p can be typed with *project* and *research project*. *project* provides two attribute definitions ordered as *budget*, *manager*, *research project* two attribute definitions ordered as *manager*, *budget*. In this example, p 's attribute order cannot be derived from the types.

We see the following solution facing these conflicts: Attributes and attribute definitions can be ordered individually. If not explicitly specified, attribute definitions are ordered alphabetically. This also applies to attributes if no type is assigned. In case of a typed content object, attributes are displayed grouped by type. Within each group the order as given by the type applies. An attribute is displayed multiple times (i.e., in multiple groups) if it is related to multiple definitions¹¹. If the content object's attribute order is explicitly defined, the order given by the definitions is ignored.

However, attributes and attribute definitions cannot be ordered in the current implementation of Hybrid Wikis. That is, the order of attributes is always computed when an individual content object is displayed. Currently the order is given alphabetically within two groups. The first group contains attributes related to definitions, the second group attributes not related to any definition. For instance, let p be a content object with three attributes *budget*, *manager*, *area* and two types (i.e., type tags) *project* and *research project*. Furthermore, let *budget* be related to an attribute definition of *project* and *area* to an attribute definition of *research project*. Then the first group contains *budget*, *area* and the second group *manager*. Determining the first group's attributes means determining the set of attribute definitions related to that object. This means, in a first step for each type tag a database query needs to be executed in order to find the set of related types. Subsequently, the keys of these types' attribute definitions are compared with the keys of the object's attributes. In case of matching keys, the attribute is shown in the first group. Even if a database query is required for each type tag in order to determine the set of related types, this is not the case for determining the related attribute definitions. This is due to the fact that the role between type and attribute definition is cached by the Tricia framework.

But even the database queries for determining the types can be eliminated. Similar to the mechanism used for caching of validation messages (cf., Section 4.2.1.6), the first group's order could be cached in the object. Then this cached order would be updated by means of Tricia batch jobs¹² in case of changes in the schema¹³. However, this mechanism is currently not implemented in Hybrid Wikis.

4.2.1.4. Storing and indexing attributes and values

Similar to type tags, attributes and values are not stored as first class entities. Otherwise, it would be required to join the database tables of content object, attribute, and value in order to get the key-value pairs of an individual object. Since this would not scale for huge sets of structured data (or even in some built-in views showing multiple attributes, such as in

¹¹Attributes not related to any definition can for example be shown in a special group *undefined*

¹²<http://www.infoasset.de/wikis/javadoc-import-wiki/platform-orm-batchdeferredchangelistener>; visited on January 13th 2012.

¹³For example, the cached order would be updated if a type's attribute definition is deleted. In this case all instances of that type would be updated.)

the type table view introduced in Section 3.4.1), we decided to use a different representation of key-value pairs in the datastore. In this situation, it is not possible to reuse the storing capabilities of the tagging module since a key-value pair cannot be considered as a simple textual tag (e.g., it is additionally required to store the data type of each value) as in case of the types. Therefore, we decided to store the key-value pairs of a content object as a JSON string. This string, representing all serialized attributes of a content object in JSON format, is stored as part of the mixin *Structured* by using a Tricia string property (cf., Figure 4.9). JSON is used as representation format since this is the primary mechanism for serializing objects (e.g., tags) in the Tricia framework.

In order to specify queries for accessing attributes with values tagging mechanisms are applied, similar to specifying queries for types. For the key as well as for each of its values, a tag is assigned to the content object using an individual namespace each (i.e., keys and values have different namespaces). In addition, tags representing values are prefixed with the key in order to conserve their connection. For instance, the key of an attribute *budget* is represented as a tag *budget* in the namespace *ATTRIBUTE_KEYS* and the attribute's value *200.000* as a tag *budget:200.000* in the namespace *ATTRIBUTE_VALUES*.

By assigning these tags (i.e., for keys and values), the Lucene index is updated accordingly. Therefore, it is possible to define exact queries [JZW09] for attributes and values by using the Lucene engine, just the same as for type tags. However, tags representing keys or values respectively are used for querying purposes only and are not visible in the UI, except for specifying queries. Finally, the tag-based querying capabilities are limited in so far that results can only be filtered according to the very attributes of content objects. It is for example not possible to express a query targeting all content objects having the attribute *owned by* set to a content object with the type tag *company*. Combining attribute, value, and type queries in conjunction is of course possible. For instance, it is possible to query for research projects which are owned by Christian Neubert.

Attributes and attribute definitions are not connected by means of a Tricia role (cf., Section 4.1.2) since attributes are serialized as JSON and not stored as first class entities in Tricia. Therefore, the determination of related attribute definitions requires to query the datastore. For instance, it is required to execute a database query in order to determine the attribute's constraints including a join operation between type and attribute definition. For a single object these database operations are less critical, but in case of multiple objects (e.g., in the type table view) this can cause performance issues. How constraint-based information is cached to avoid performance problems is explained in Section 4.2.1.6.

Even if representing attributes as entities and connecting them to corresponding attribute definitions by means of a Tricia role, it would be laborious to keep this role valid, since the Tricia framework uses caching mechanisms for roles. For instance, if a type is renamed all roles from its attribute definitions to attributes need to be updated and additionally the role-caches have to be invalidated accordingly. This operation cannot be performed synchronously since potentially a huge set of content objects is affected. But asynchronous execution potentially leads to inconsistencies in the datastore for example in case of concurrent access when using a role which is not yet invalidated. Therefore, we consciously decided to separate concepts representing the data (i.e., attributes, type tags) from the schema (i.e., types, attribute definitions, and constraints) and query schema information (or data information respectively)

always on demand. However, as discussed before representing attributes as entities is not an option anyway.

The process of assigning an attribute including serialization and indexing steps is exemplified in Figure 4.10. This figure additionally illustrates how the validation messages for an individual attribute are determined and stored (cf., Section 4.2.1.6).

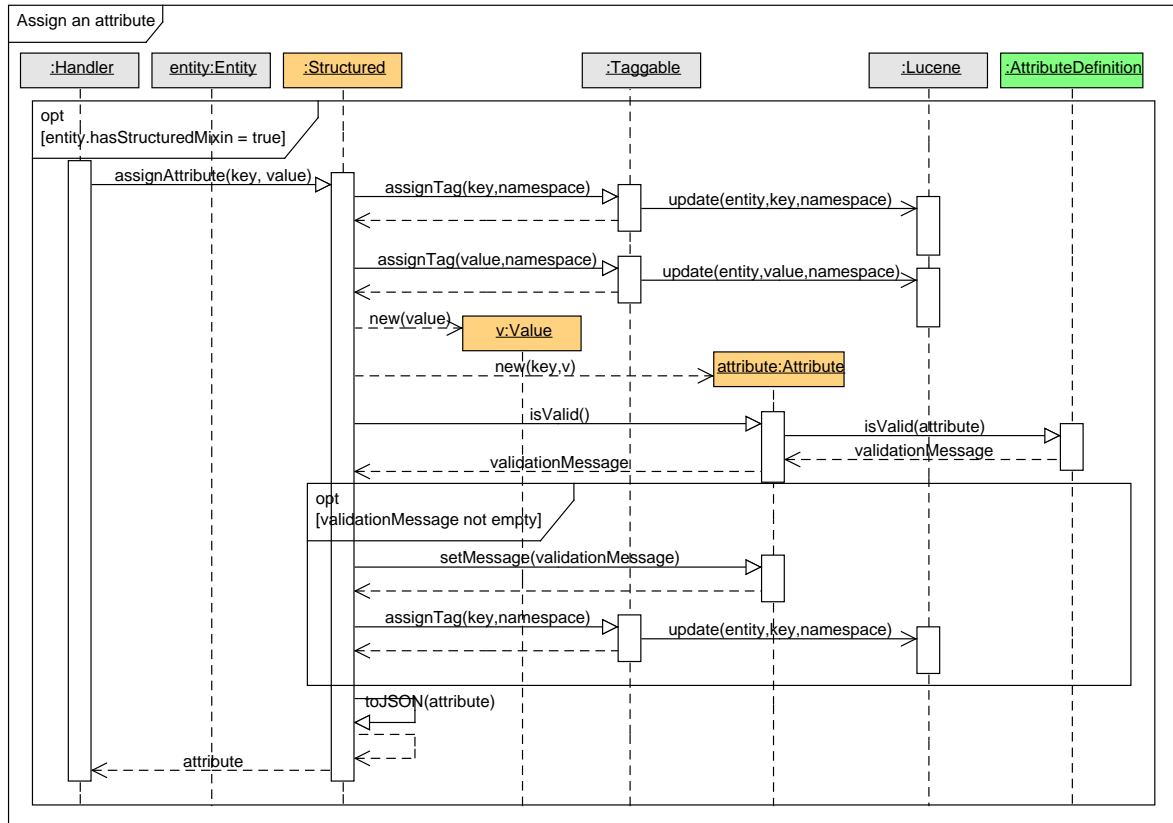


Figure 4.10.: Storing and indexing of key-value pairs.

4.2.1.5. Implementation of data types

As explained in Section 4.2.1.1, values are not stored as first class entities due to reasons of performance. In particular, this has an impact on structured link values, since in this case structuring support cannot directly be applied by using a mixin. Therefore, a structured link value is additionally associated with a first class entity that supports structuring capabilities by means of a mixin (cf., *LinkStructure* in Figure 4.11). By doing so, the link's types and attributes are indexed by means of Lucene as in case of structured wiki pages. Furthermore, in this way it is possible to regularly specify queries regarding the link structures (i.e., in combination with fully fledged structured entities, such as wiki pages). For instance, a search result based on a query asking for objects having a specific type potentially contains both, structured wiki pages and structured links. Using an additional entity as the primary store of structured links also has the advantage that further mixins can be applied. Therefore, from a

technical point of view, it is convenient to support search and link capabilities for structured links by means of corresponding mixins (cf., Figure 4.11).

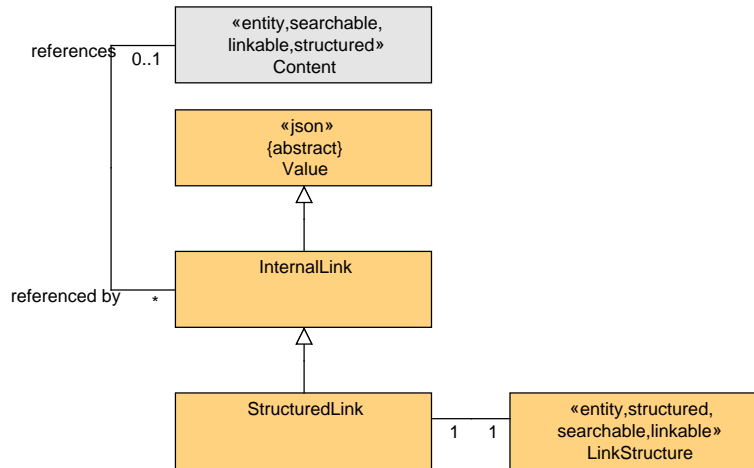


Figure 4.11.: Link structures are represented by an entity with mixins

Using an additional entity of course requires executing a database query in order to access the structured part of the link. Therefore, we decided to display structured links as normal links in the UI only indicating that additional information is contained but not retrieve it from the database in order to reduce the number of queries. That is, the attributes and types of a structured link are never shown per default, only if a user explicitly requests to display them. This is particularly important when displaying multiple values in a single view, such as the type table view. One way to access the structures attached to a link is using the built-in link view as introduced in Section 3.4.1.4.

The value representing the link (i.e., the internal reference to the content object) is serialized as JSON as the other values. It is also conceivable to serialize the attributes and types instead of using a separate entity. However, in this case linking and searching is not supported built-in by the Tricia framework.

Finally, we implemented the data types record (cf., Section 3.1.5.5) and hypertext (cf., Section 3.1.5.4) in a prototype only, that is, they are not yet applied or evaluated in practice (cf., Chapter 6).

4.2.1.6. Validation

As introduced in Section 3.1.8, the simplest form of indicating constraint violations for individual content objects is to determine related attribute constraints on demand (e.g., when displaying a content object) and apply them to the current attribute values in order to generate a validation message. Potentially, this reduces the comfort for users when reading a content object. For example, when a content object has multiple attributes related to attribute definitions from different types calculating all messages potentially need some time. Therefore, the validation messages are directly stored as part of the attributes (cf., Figure 4.6).

By doing so, no additional database queries are required to determine the validation messages for individual content objects. Thus, an attribute can be considered as a cache providing fast access to some schema-based information. However, this requires to keep the cached validation messages up-to-date.

As shown in Figure 4.10, when assigning a key-value pair the validation messages are calculated and in case of invalid values (initially) set in the attribute. Additionally, cached messages need to be updated in case of changes to the content object, in particular changes to:

- Values, since a changed value can violate or meet related constraints.
- Type assignments, since with a changed set of types it is possible that constraints are not related to attributes anymore or new ones apply to them.
- The content object's space, since it is possible to move the content object to a different space. In this case, other constraints from different types apply to the object.

Furthermore, updating the validation messages is required in case of changes to the schema, in particular changes to:

- Types, since when changing a type content objects possibly lose their connection to this type. For instance, renaming a type is not necessarily propagated to the related instances (cf., Section 3.3.3).
- Attribute definitions, for the same reason as for changing types.
- Constraints, since it is not required to consolidate related values when changing a constraint. Additionally, it is even not possible to propagate changing constraints to all instances in some cases (cf., Section 3.3.3).

In case of changes to the content object, the set of related constraints need to be determined in order to update the current validation status of the object. In case of changes to the schema, the set of the affected content objects needs to be identified for updating their status accordingly. Since the determination of related elements as well as the update of all attributes' validation messages potentially needs some time, especially in case of types with many instances, both is performed asynchronously by using Tricia batch jobs. By doing so, the caching of validation messages helps to ensure fast access to content objects thereby facilitating reading of contents, since no additional database queries are required in order to calculate the current validation status. But also changes in the schema (or in the data respectively) requiring updates of the cached validation messages do not slow saving operations from a user's perspective because of using asynchronous batch jobs. Since a batch job potentially takes some time attributes can contain obsolete validation messages. In such cases, related content objects show a wrong validation status. In particular, this is a drawback when explicitly search for constraint violations (cf., Section 3.1.8). However, the delayed execution of batch jobs never leads to inconsistencies in the data or the schema.

As discussed in Section 3.1.8, it is important that it is possible to explicitly search for constraint violations (e.g., since the validation status of attributes can change without modifying the underlying content object). Therefore, the current implementation supports two possibilities in order to find them:

- The search for invalid content objects. This means at least one attribute of the object has invalid values.
- The search for attributes (i.e., attribute keys) having invalid values.

As indicated in Figure 4.10, this is also realized by assigning tags with a specific namespace. By doing so, it is possible to query these tags by means of the Lucene search engine and additionally to combine queries regarding the validity of objects with queries regarding types (i.e., type tags) and/or attributes (i.e., attribute tags). For instance, if a project is constrained to have at most one project manager users can search for research projects which are managed by Christian Neubert and provide more than one manager. However, the user needs to manually find out which constraint exactly causes the violation (cf., Section 3.2.4.1). In order to be able to search for content objects having at least one invalid attribute the object is tagged as invalid within a special namespace. Another namespace is used to tag invalid attributes. This works similar as for the attributes' values, that is, using the attribute key as a prefix in order to distinguish the invalid attribute flags of a content object.

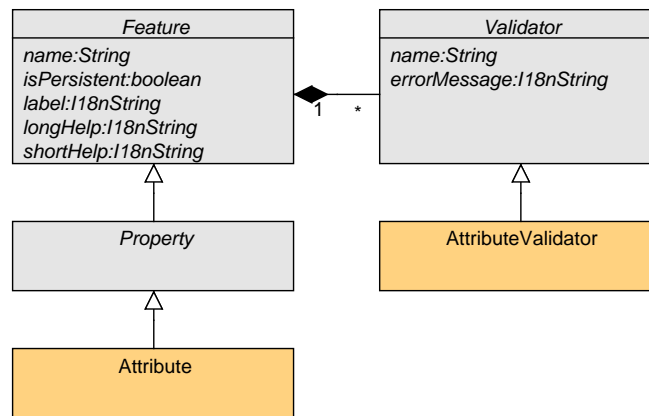


Figure 4.12.: Attributes and constraints integrated in the Tricia data model.

As discussed in Section 3.1.8, when using strict or rigid types users are not allowed to save a structured object containing any invalid attributes. In order to ensure this the Tricia validation model needs to be extended. We decided to implement attributes as subtypes of Tricia properties in order to integrate them in the existing validation model (cf., Figure 4.12). This is also because attributes and built-in properties are quite similar, that is, both can be considered as key value-pairs. However, built-in properties cannot dynamically be changed (i.e., new properties cannot be added to or existing ones be removed from an object) at runtime as it is possible for attributes. Due to the deep integration in the framework, attributes in Hybrid Wikis can be treated as normal Tricia properties. In particular, this is necessary since the validity of each feature (roles and properties) is checked by means of Tricia validators when saving an entity (cf., Figure 4.13). Therefore, an entity is extended in so far that the set of its features includes both, attributes and built-in properties (cf., Figure 4.14). These two kinds of features are distinguished by using a specific key, that is, attribute keys are single-quoted when using them for programming purposes. For instance, the key *content* refers to a built-in property, the key *'content'* to an attribute.

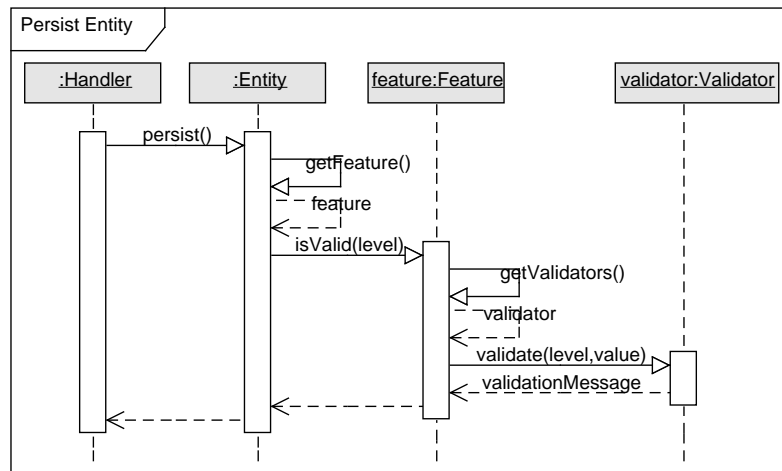


Figure 4.13.: Tricia's validation model checking the consistency of entities with properties and roles.

The extended validation model is depicted in Figure 4.15. This representation is simplified. Of course the validity of all features is checked when saving an entity. However, in case of an attribute the validity is checked by using its cached validation messages. In this situation, the attribute validator ensures that a validation message is only returned¹⁴ if the attribute is related to a strict attribute definition. In order to determine if an attribute is related to a strict attribute definition a database query is required. If persisting an entity results from a user interaction, this query cannot be executed asynchronously since the user immediately needs feedback in case of failing validators. This can be optimized by caching the strict flag within the attributes, similar to the validation messages. This means that the cached flag needs to be updated when changing the schema. For instance, when declaring a constraint as strict the flags of all related object needs to be updated accordingly. But this would allow to enter invalid values for a certain period of time. This is due to the fact that updating of all related objects potentially needs some execution time and the Tricia framework does not prevent concurrent data access by providing locking mechanisms. Therefore, caching of the strict flags is currently not implemented.

As indicated in Figure 4.15, the Tricia framework supports different validation levels. These levels are necessary to control the granularity of the validators¹⁵. Basically, two kinds of levels are supported. The first level prevents writing invalid values to the datastore, the second level prevents users from submitting invalid entities in the UI. Attribute validators only make use from the latter. That is, only in the UI it is permitted to submit entities with invalid attributes and only in case of strict constraints. This also means that entities can be stored to the database when processing entities programmatically although they contain invalid attributes and even in case of strict constraints. For instance, the ability to ignore validation rules on the programming level is especially important when importing structured data from external datasources (cf., Section 4.2.6) or when interacting with external systems

¹⁴Returning an empty validation message indicates the validity of a content object

¹⁵<http://www.infoasset.de/wikis/javadoc-import-wiki/platform-orm-validator-validationlevel>; visited on January 13th 2012.

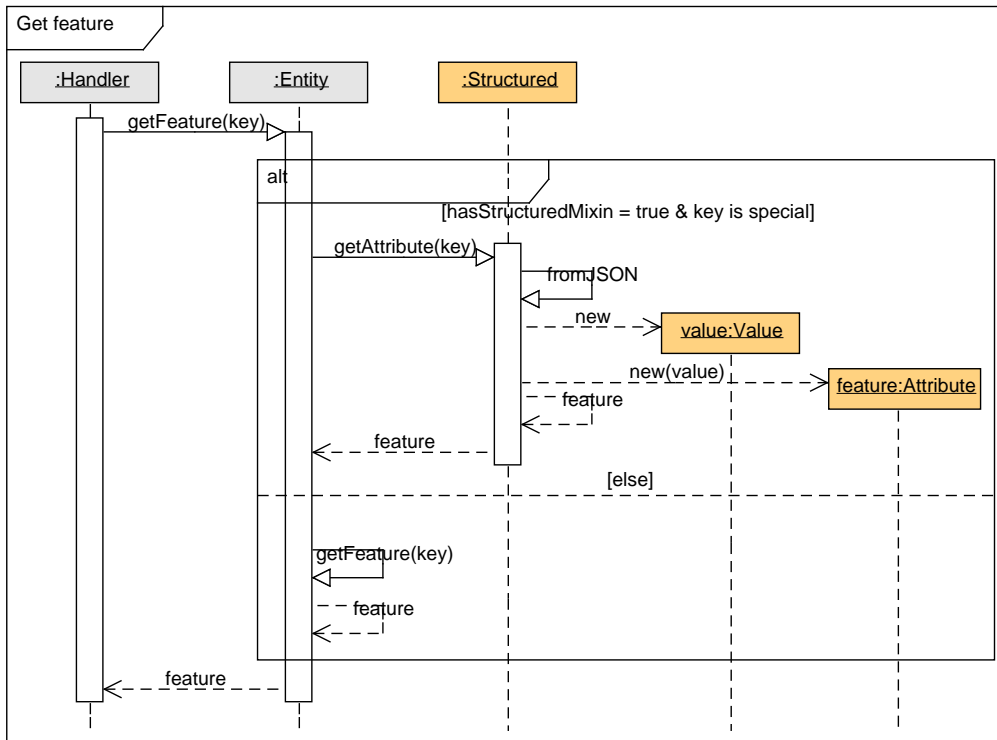


Figure 4.14.: Attributes integrated in the Tricia data model.

(cf., Section 4.2.7). However, the validation of built-in features can still prevent persisting an entity (e.g., in case of ambiguous wiki page names within the same space).

Besides the smooth integration in the validation model, another advantage of implementing attributes as part of the Tricia data model (cf., Figure 4.12) is that it is very convenient to build advanced controls representing structured information in the UI¹⁶ from a programming perspective. This is facilitated since attributes are fully integrated in Tricia’s data modeling framework (cf., Figure 4.12).

4.2.1.7. Value formats

In an early version, Hybrid Wikis supported the two data types string and link only in order to keep it simple from both perspectives, the end-users’ and the software engineer’s. However, in an enterprise environment additional data types are required, such as number and date. In particular, it is important that values are represented appropriately in all views. For instance, the values *1.3*, *100.3* would be aligned left in case of storing them as plain strings. But since they are likely to be numbers it would better to align them right in order to show the decimal points vertically aligned to each other. Additionally, it is important that values can be sorted

¹⁶<http://www.infoasset.de/wikis/javadoc-import-wiki/platform-documentation-functionsubstitut>; visited on January 13th 2012.

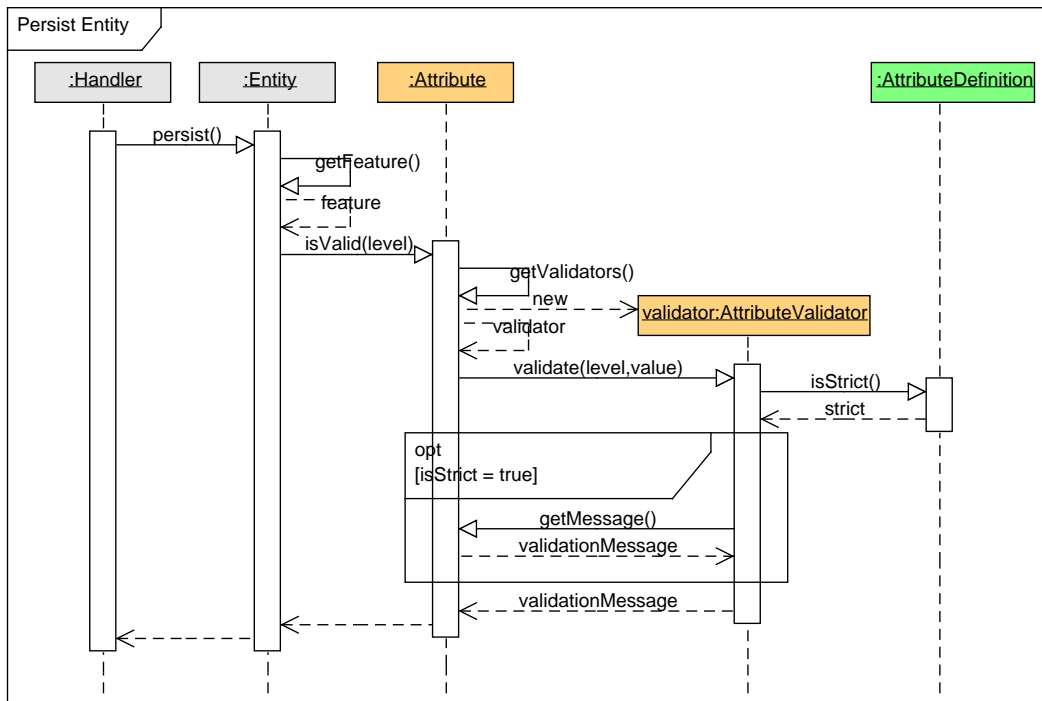


Figure 4.15.: Tricia's validation model extended with attributes and constraints.

correctly. For example, the values *12* and *1* are sorted incorrectly when storing them as strings.

In [Ma11] Hybrid Wikis are extended with additional data types, such as number and date. The research questions underlying that thesis are:

- How can users be supported when interacting with values and data types in Hybrid Wikis?
- How can values and data types be stored and displayed in Hybrid Wikis?

The following findings result from responding to these questions according to [Ma11]. The general process of the values' interactive management is depicted in Figure 4.16. First, a user enters a value with data type and submits it to the server. Then, the value is transformed into an internal representation according to the given data type and stored as a JSON string in the database. This string contains a canonical representation¹⁷ of the value and some additional information required for formatting the values correctly in a view, such as the precision of a number. The conversion into a canonical representation potentially results in a reduction of the originally entered value's accuracy. That is, the value is potentially represented differently when it is edited later. Since this might be confusing for users, a preview could be shown indicating how the value is displayed when it is viewed or edited later. However, such a preview is currently not implemented.

¹⁷<http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>; visited on January 17th 2012.

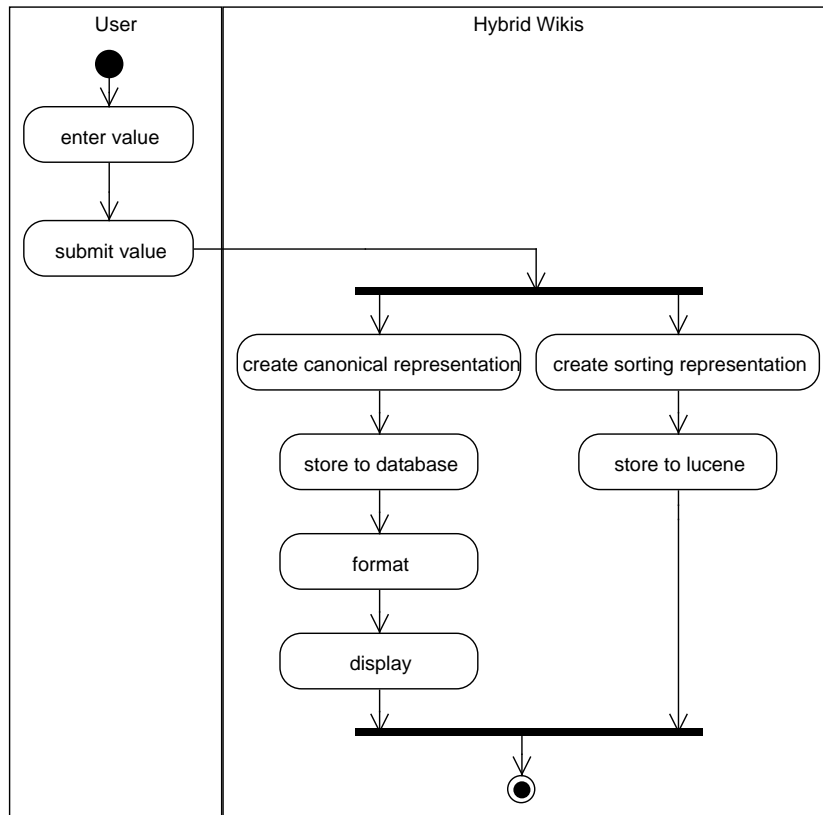


Figure 4.16.: Interactive management of values with data types.

Additionally, the Lucene index is updated twice¹⁸. The first update is necessary in order to define exact queries as described in Section 4.2.1.4, the second in order to efficiently sort values correctly in the UI. In particular, this is necessary in the tabular overview of a type tag (cf., Section 3.4.1). A column (representing a key) potentially contains values of different data types in this view. Since the query underlying this table's data is based on the Lucene search engine we decided to also use its sorting capabilities to sort individual columns in ascending or descending order. Otherwise, it would be necessary to sort the query results programmatically in a second step, which is more complex from a software engineering perspective.

Since Lucene's sorting algorithm relies on a lexicographic order each value is indexed in a canonical lexical representation additionally prefixed with its data type. By doing so, the values are grouped per data type and sorted within each group according to the canonical representation. Additionally, the data type prefix determines the order of the groups. We selected the prefixes in such a way that the group's order is given as string, date, number, link (internal, external), structured link, record, and hypertext. For instance, a string value would be represented as *a.value* and a date value as *b.value*. In case of multiple values per attribute, the data type (i.e., the prefix) of the first value is decisive for determining the group of the attribute. Additionally, the first value's canonical representation determines the rank within

¹⁸In Figure 4.16 only one Lucene update is depicted for purpose of illustration.

a group in case of multiple values per attribute. For example, let p_1 and p_2 be content objects typed with *project*. Both provide an attribute *member*, p_1 with the string values *Berta*, *Dora* and p_2 with the string value *Chris*. When the type table (cf., Section 3.4.1.5) of *project* then is sorted in ascending order by *member*, both *member* attributes are contained in the same group (i.e., both attributes are prefixed with *a:*) and p_1 is listed before p_2 since the second value of p_1 's *member* attribute, *Dora*, is ignored.

In the UI the value is displayed formatted according to the viewing user's language after saving it to the database and updating the Lucene index. For formatting purposes the data model is extended as depicted in Figure 4.17. For each data type a corresponding formatter is implemented.

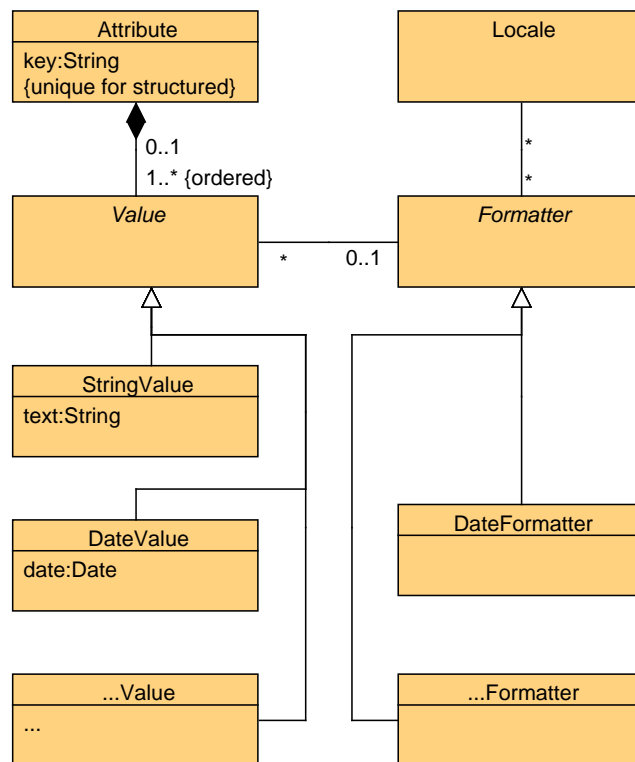


Figure 4.17.: Formatting values according to data type and user language.

For example, the different representations of a date are listed in Table 4.2. This example assumes that the value is edited and viewed by different users, each with different locale. Otherwise, the displayed value would also be *06.06.2012*.

But it is also possible that a value's representation differs for the same user in both contexts (i.e., edit and display). For instance, a number displayed as *12.23* can be represented as *12.231* when it is edited [Ma11]. That means the edit representation is the most precise human-readable representation of a value in Hybrid Wikis. Therefore, this representation is also used for the interpretation of values as discussed in Section 3.3.

The implementation of the thesis [Ma11] provides further outcomes:

- Data type suggestions when editing values (cf., Section 3.3.1.2).

Table 4.2.: Internal and external representation of values according to [Ma11].

Context	Representation
User input	06.06.12
Database	1338933600000
Lucene (sort)	d:1338933600000
Edit	06.06.2012
Display	12/06/06

- Advanced data types, such as currency and percentage.
- Alignment of values according to their data types (e.g., a number value is aligned right when it is displayed).

The author also discusses different possibilities to show validation messages while editing a value. However, the provided solution only indicates errors in case of using strict constraints. Therefore, the current implementation is adapted as depicted in Figure 4.18. In our solution, invalid values can only be stored if they are not strict (cf., warning in Figure 4.18). Additionally, validation failures are indicated while editing values, that is, it is not required to store an attribute in order to show constraint violations to the user. Further insights regarding the implementation of different data types in Hybrid Wikis can be found in [Ma11].

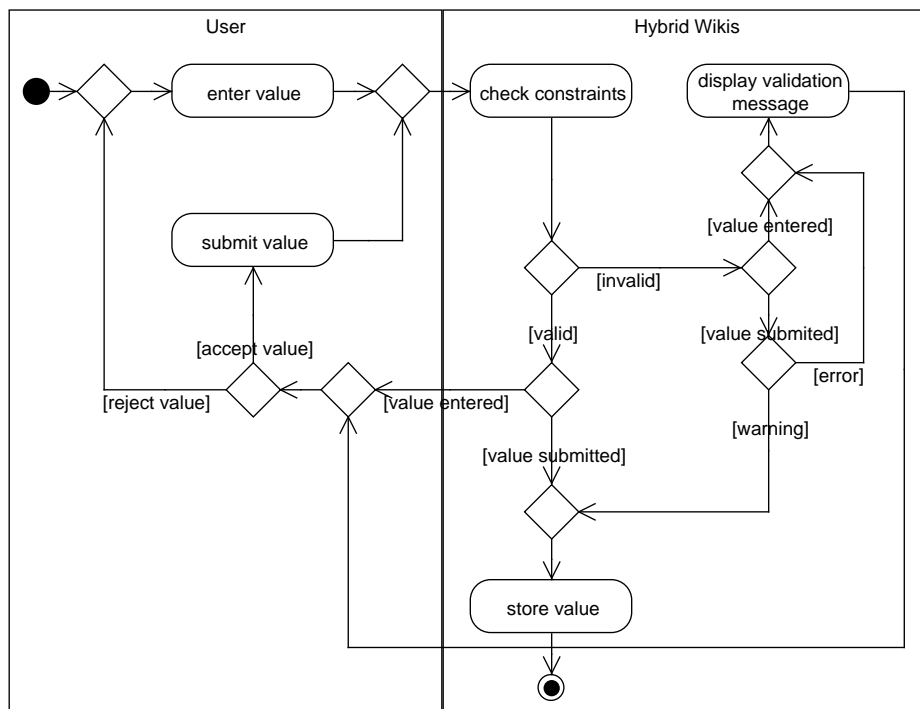


Figure 4.18.: Validation while editing values.

4.2.1.8. Types, attribute definitions, and constraints

Types and attribute definitions are realized as Tricia entities supporting link and search capabilities (cf., Figure 4.6). Additionally, both are protected from unauthorized access by applying the access control list from the owning space.

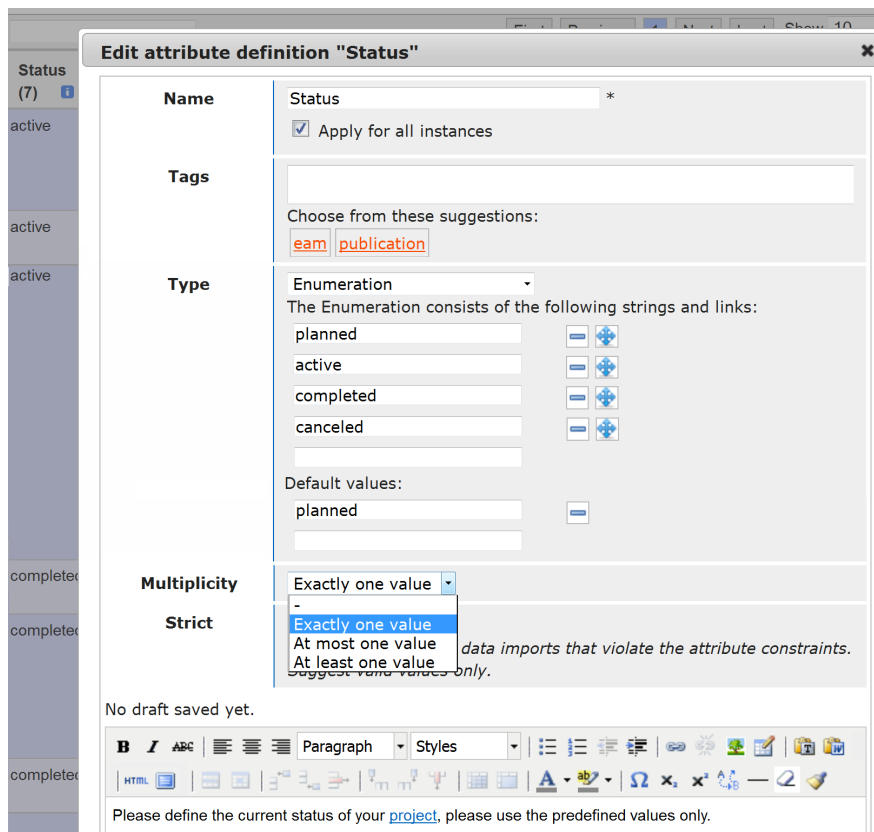


Figure 4.19.: Attribute definition with constraints.

As depicted in Figure 4.6, instead of using Tricia entities constraints are stored as simple JSON strings as part of their owning attribute definitions. This is

- due to reasons of performance, since otherwise it would be necessary to perform database joins in order to determine the attribute definition's constraints, and
- it is not required that constraints can be extended by mixins (e.g., in order to make them searchable).

For reasons of simplicity the current implementation of Hybrid Wikis only supports three kinds of multiplicities represented as simple domain values (cf., Figure 4.19). This means that it is not possible to explicitly specify lower and upper bounds of a constraint (cf., Section 3.1.8). The following values are supported:

- Exactly one value (1..1)
- At most one value (0..1)

- At least one value (1..*)

This is not due to technical restrictions. We consciously decided to provide only the most common multiplicities to end-users.

4.2.2. Extended architecture

The extended architecture of Tricia is depicted in Figure 4.20. Hybrid Wikis provide two additional plugins. The first plugin, namely structured, is directly built on top of the Tricia core in order to support information structuring for all kinds of content objects in higher levels of the plugin stack, such as wiki pages in the wiki plugin or blog posts in the blog plugin. However, the core plugin also provides entities having unstructured content (e.g., persons and groups). In order to enable structuring of these objects all elements necessary for structuring are directly integrated in the core plugin within a separate package. That is, the structured plugin is part of Tricia's core plugin. However, both plugins are separated in Figure 4.20 for purpose of illustration.

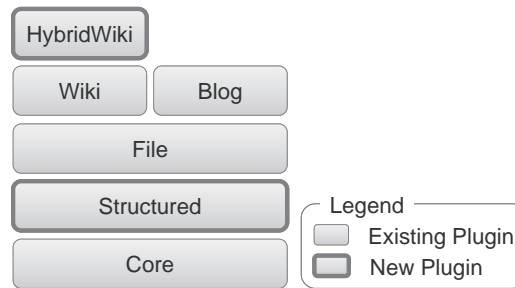


Figure 4.20.: Plugins provided by Tricia including Hybrid Wikis.

The structured plugin consists of additional entities, views, and handlers (cf., Figure 4.21). The additional views and handlers primarily serve for the generation of the built-in views (cf., Section 3.4.1). Which of the concepts introduced in Section 3.1 are represented as Tricia entities is discussed in Section 4.2.1.

Individual content objects can access elements within the structured plugin (e.g., the mixin *Structured*, attribute definitions, types) in order to support information structuring. The current implementation of Hybrid Wikis supports structuring for

- wiki pages,
- files,
- persons, and
- groups.

The entities file, person, and group can be structured by default, that is, they directly make use of the mixin *Structured*. Wiki pages can only be structured when using an additional plugin, namely hybrid wiki. This is due to the fact that in some cases it is required that wiki

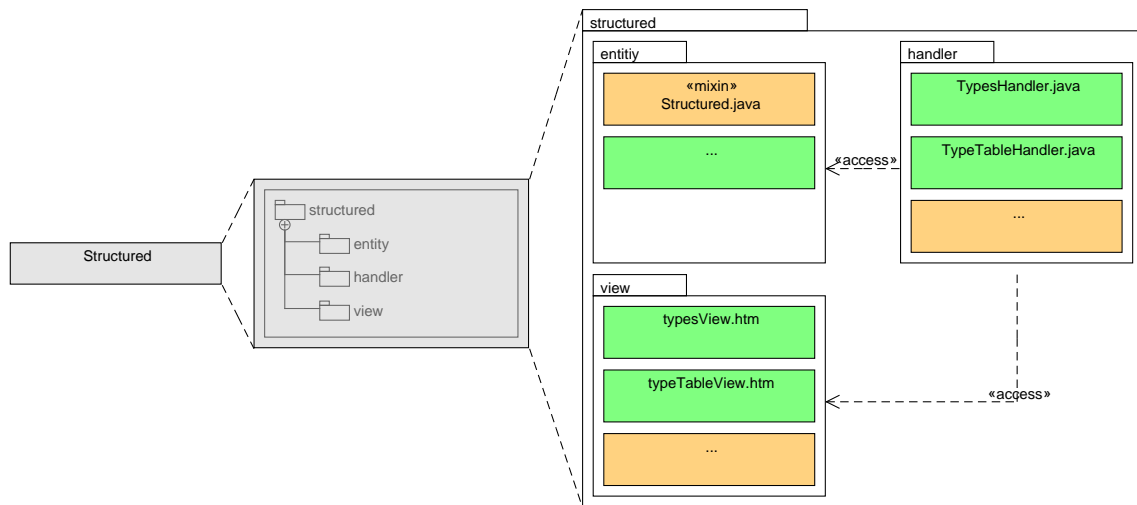


Figure 4.21.: Constituents of the structured plugin.

pages only consist of plain unstructured content¹⁹. Therefore, the ability of wiki pages to be structured is encapsulated in a separate plugin on top of the wiki plugin. By doing so, it is possible to activate structuring of wiki pages for different Tricia installations selectively.

The hybrid wiki plugin uses the extension mechanisms provided by the Tricia platform (cf., Section 4.1.5) in order to extend the wiki page entity by using the mixin *Structured* and the wiki page views by showing for example the structured box (cf., Section 3.4.1.2) with attributes, type assignments, and inverse links. The constituents of the hybrid wiki plugin are sketched in Figure 4.22.

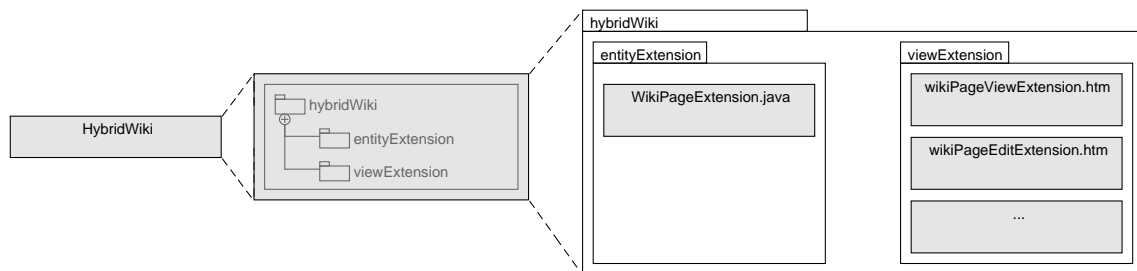


Figure 4.22.: Constituents of the hybrid wiki plugin.

¹⁹It is planned to investigate how users interact with Hybrid Wikis compared to traditional wikis (i.e., without structuring support) in industrial case studies.

4.2.3. Suggestions

In Section 3.3.1, we introduced suggestions as lightweight means to encourage users to provide information structures. In the following, we explain which of these suggestions are currently provided by Hybrid Wikis and how they are realized from a technical point of view.

4.2.3.1. Attribute suggestions

As described in Section 3.3.1.1, attribute suggestions of an individual content object originate from its structural similarity to other objects. Types, tags, and attributes are considered when determining the similarity. Additionally, the keys of attribute definitions are suggested, but only if the object is related to any. Therefore, determining suggestions is currently implemented in two steps. In the first step, the related attribute definitions are calculated using a database query.

In the second step, similar content objects are determined by means of the Lucene index. The types, tags, and attribute keys of the current object are combined in a disjunctive query. That is, it is searched for the attribute keys (cf., Section 4.2.1.4) of other content objects providing the same or most of the object's current types, tags, and attributes. For this purpose Lucene's built-in mechanisms for determining the similarity of resources²⁰ is used. These mechanisms also consider if resources have multiple types, tags, or attributes. This means, for example attributes are preferred if they occur for content objects having all or several of the current types assigned over attributes for objects with no types. However, in this way types, attributes, and tags are not yet prioritized among each other. That is, as proposed in Section 3.3.1.1 attributes resulting from the similarity of objects with multiple types should have a higher priority than attributes resulting from the similarity of objects with multiple tags. Therefore, the individual query parts are additionally boosted²¹. For instance, the part of the query representing the types is more boosted than the tags part. In this way, it is possible to influence the search results according to the following ranking: types, tags, attributes. The most frequent attributes resulting from that combined, boosted queries are suggested.

Finally, the results from both queries (i.e., database and Lucene) are merged together considering that suggestions resulting from attribute definitions have the highest priority. Besides attributes, the other suggestions introduced in Section 3.3.1 are realized as follows.

4.2.3.2. Autocompletion

Autocompletion for attribute keys works similar to calculating attribute suggestions. The main difference is that the query additionally only filters keys starting with the partial string entered by the user (cf., Section 3.3.1.4, Figure 4.23). The search for prefixes (e.g., the partial string of an attribute key) is efficiently supported by means of the Lucene index²².

²⁰http://lucene.apache.org/java/2_9_0/api/all/org/apache/lucene/search/Similarity.html; visited on January 29th 2012.

²¹http://lucene.apache.org/java/3_5_0/scoring.html; visited on January 29th 2012.

²²http://lucene.apache.org/java/3_0_0/api/all/org/apache/lucene/analysis/TokenStream.html; visited on January 29th 2012.

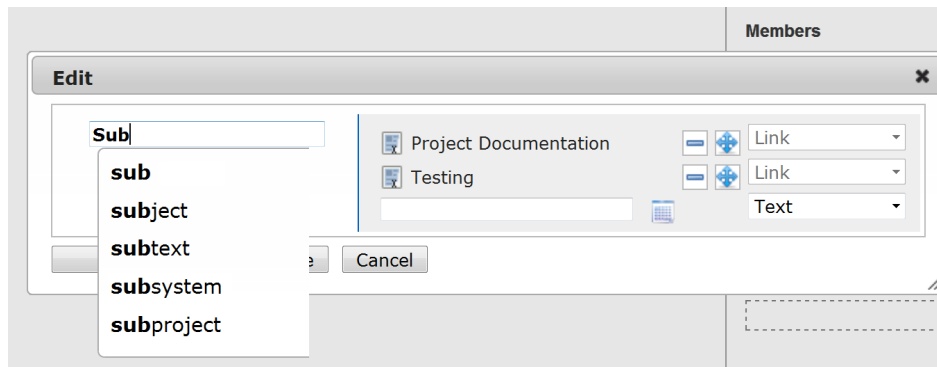


Figure 4.23.: Autocomplete suggestions for attribute keys.

Basically, users enter values using a simple input field (cf., Figure 4.24). Autocompletion is provided when they start typing (cf., Figure 4.24 (1)). Then the autocomplete control presents a mixture of textual values (cf., Figure 4.24 (2)), such as strings and numbers, and links (cf., Figure 4.24 (3)). Even if links have a textual representation it is necessary that users can distinguish them from simple values. Currently the autocomplete control is limited to five visible hits. This seems very restrictive. However, in [SS11] the authors show “that users prefer to refine their query rather than scanning the complete list of search results”. That is, users change the partial search string until the expected value is shown within the most frequent hits, even if that value is suggested before in a lower position.

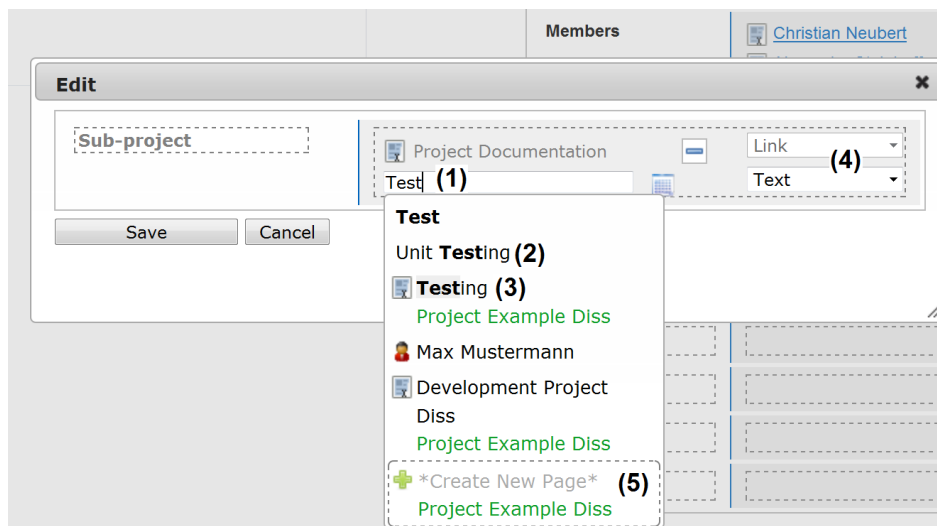


Figure 4.24.: Autocomplete suggestions for attribute values.

Values shown in the autocomplete control are determined in three prioritized steps. Each step is only executed if the given limit of five hits is not yet exceeded. We consciously decided to prioritize these steps in order to prefer values originating from constraints.

The first step determines matching values according to related constraints. These values are additionally filtered by the partial string given by the user. For instance, if an attribute is related to an enumeration constraint (cf., Section 3.1.8.6) the values specified for this constraint

(i.e., its enum values) are provided as autocomplete results, but only values starting with the partial string the user has entered (cf., Figure 4.27). The related constraints are determined using a database query. In case of a link constraint, link values are suggested filtered by matching link target types (cf., Section 3.1.8.8). In this case, an additional Lucene access is required using an exact query according to the constraint's link target types. That is, it is searched for content objects having at least one of the given link target types. The filtering of matching values according to the partial string is currently performed in memory (e.g., the enum values or the links resulting from the Lucene query).

The second step determines matching textual values, such as strings and numbers. This works similar to the autocomplete of keys, that is, the search hits are filtered by values starting with the partial string only. The difference is that the search context is additionally restricted to values used in combination with the current attribute key in this case. That means values occurring together with the current key are preferred in the search. Textual values for the autocomplete of values are determined using the Lucene index.

The third step determines links to other content objects. For this purpose Tricia's built-in autocomplete mechanism is utilized which is also based on the Lucene index. The underlying query considers the partial string within the object's built-in content, tags, name, attributes, and values in order to determine the set of relevant link hits. Therefore, it is possible that a link is suggested even if the partial string is part of an attribute having a different key than the current attribute.

4.2.3.3. Data type suggestions

Data type suggestions are provided as explained in Section 4.2.1.7. That is, a data type is suggested when the user enters a value. As depicted in Figure 4.24 (4), a data type is presented as a combobox. When a user starts typing this box changes to a data type recommended based on the entered value. Additionally, if the user selects a data type which is rarely used by other attributes in a similar context (i.e., with the same key and the same type(s)) this is indicated in the UI (cf., Figure 4.25 (2)) and users can display the distribution of data types on demand (cf., Figure 4.25 (1)). These mechanisms encourage users to follow the implicit schema.

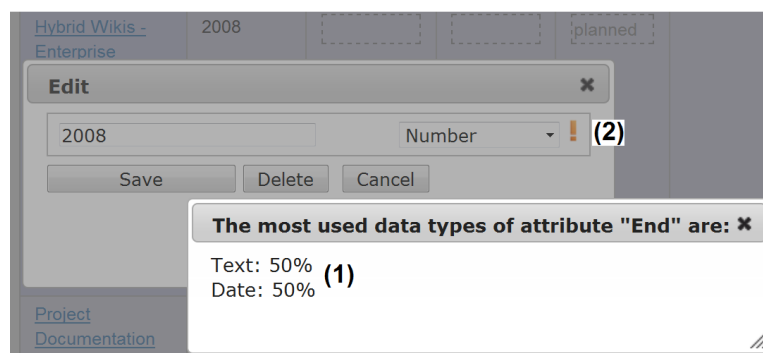


Figure 4.25.: Data type suggestions are provided while entering values.

4.2.3.4. Type suggestions

The realization of type suggestions is completely based on the tagging module and its UI controls. That means type suggestions are provided when a user changes the type tags of a content object (cf., Figure 4.26 (1)). The tagging controls also provide autocomplete support for type tags built-in (cf., Figure 4.26 (2)).

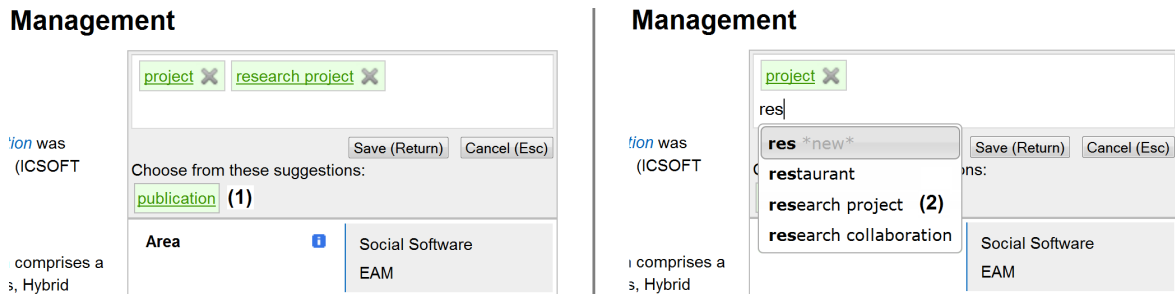


Figure 4.26.: Suggestions and autocomplete support for types.

4.2.3.5. Constraint-based suggestions

As discussed in Section 4.2.3.2, constraints directly impact the autocomplete behavior since for example values of an enumeration constraint are preferred in the result list (cf., Figure 4.27 (1)). This way, users are urged to choose constraint-based value suggestions preferably (cf., Section 3.3.1.5). A data type is suggested if an attribute is related to a data type constraint. For instance, a user decides to fill out a suggested attribute. Then this attribute provides a corresponding key in the edit dialog (cf., Figure 4.27 (3)). In this case, the data type from a related data type constraint is given as default (cf., Figure 4.27 (2)).

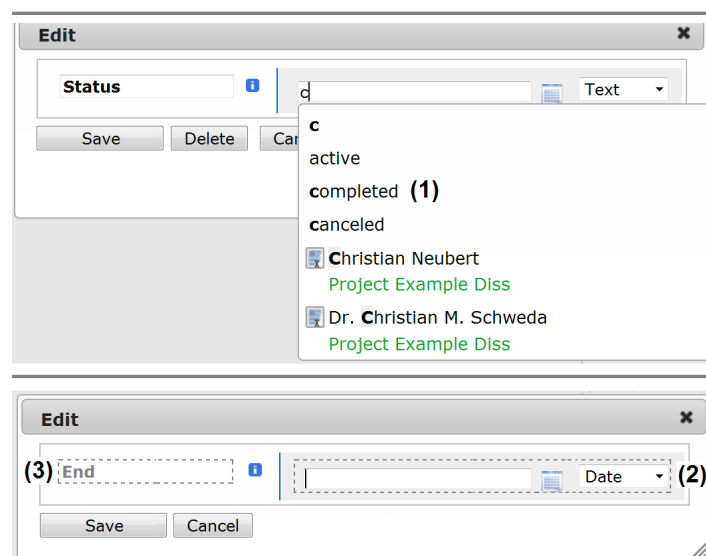


Figure 4.27.: Improved input support for values based on attribute definitions and constraints.

Furthermore, constraint-based suggestions are realized using advanced UI controls. Values specified in an enumeration constraint are provided in a drop-down list if a user clicks in the value input field (cf., Figure 4.28 (1)). In case of a date constraint, a date picker is offered when clicking in the value field.

Attributes related to a multiplicity constraint show a corresponding number of input fields if they are edited. In case of 0..1 and 1..1, exactly one input field is given. However, even if only one field is shown, users are not prevented to enter multiple values, except when using strict constraints. In case of 1..* (or if no constraint is given), an empty field is shown allowing users to enter an additional value (cf., Figure 4.24 (1)).

Default values are shown as part of attribute suggestions. That is, if an attribute suggestion is related to a constraint the defaults are shown as its values (cf., Figure 4.28 (2)). By doing so, it is very convenient for users to choose one or more of these values when filling out an attribute suggestion.

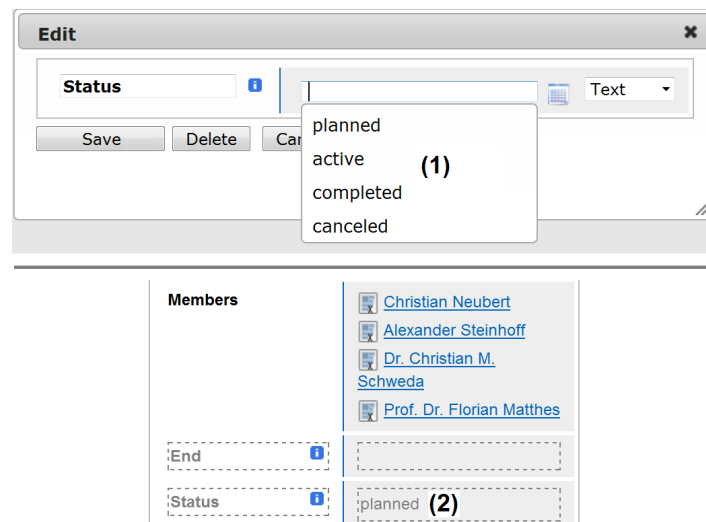


Figure 4.28.: Facilitating data entry based on attribute definitions and constraints.

4.2.4. Transitions

Hybrid Wikis currently provide the following transitions supported by drag and drop:

- Transfer unstructured content to attributes (cf., Figure 4.29).
- Merge attributes.
- Transform a tag into a type tag.

In the first case, users can decide to create a new attribute or to merge the transferred value into an existing one. In both cases, the data type is guessed and suggested to the user. For instance, if the transferred value can be interpreted as a number the data type number is suggested. Furthermore, transitions between data types are supported by manually changing

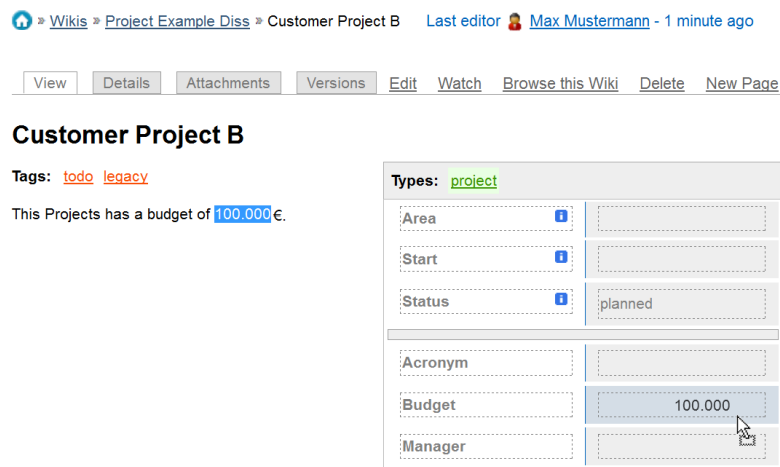


Figure 4.29.: Transferring unstructured content to an attribute.

the combobox depicted in Figure 4.24 (4). The combobox only contains data types from executable transitions.

The objectification of structured values (cf., Section 3.3.2.4) is integrated in the autocomplete control. The control always provides an additional entry at the end of the autocomplete suggestion list (cf., Figure 4.24 (5)). When choosing this entry a new wiki page is created having the current partial (search) string as the name, for example in Figure 4.24 a new wiki page with name *Test* is created. If the attribute is related to a link constraint specifying default types, these types are additionally applied to the new page. For example, if the attribute *Sub-project* (cf., Figure 4.24) is related to a link constraint having the default type *project* then the page *Test* is additionally typed with *project*. This way, information is structured without burdening users with additional, unnecessary effort. Furthermore, users are encouraged to also structure the new page since it is likely that attribute suggestions are provided resulting from these additional types. Additionally, in this way extracted links are valid automatically since they conform to the target types of the link constraint. This makes it very convenient for users to transform textual values into links (and objects) while entering data. However, the objectification of an existing value currently is only possible with the help of the autocomplete control.

Objectification of semi-structured content (cf., Section 3.3.2.5) is currently supported for simple hypertext tables (cf., Figure 4.30 (1)). A table (cf., Figure 4.30 (2)) is transformed into structured wiki pages having the first row's cell as name per convention (cf., Figure 4.30 (3)). This means that it is not possible to use a combination of multiple columns as the page's name as proposed in Section 3.3.2.5. Furthermore, it can be specified which types to assign to the newly created pages. In case of already existing pages, the structure is updated accordingly²³. For instance, if a page already exists having an attribute with a key corresponding to a column header the cell's value is merged with the values of that attribute. In order to make the merging of values more understandable for users and to show them which wiki pages are updated or newly created we implemented a preview dialog. This way, users can check and

²³If a value of a cell in the first row matches the name of an existing page, this page's structure is updated. No new page is created in this case.

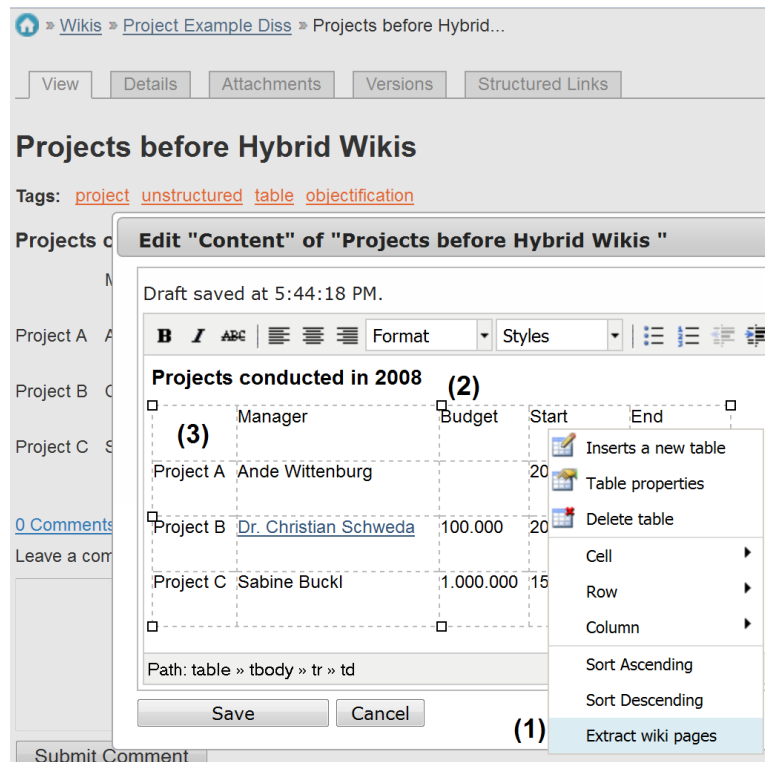


Figure 4.30.: Objectification of a semi-structured table from the built-in content of a wiki page.

adjust the structures (e.g., change a page name or add additional types) before executing the transition.

In order to support the objectification of unstructured content (cf., Section 3.3.2.6) we adapted the Tricia autocomplete control used to insert links in the built-in content. We enhanced this control with a special entry. Similar to the objectification of structured values, this entry allows to transform a text value into a new wiki page having the text as its name. Additionally, the original text is replaced by a link to that page.

4.2.5. Consolidation

Consolidation techniques are currently only realized as introduced in Section 3.3.3. In particular, changes to types and attribute definitions can be applied to related instances, that is, users can decide

- to change the type tags of related content objects when renaming a type (cf., Figure 4.31 (1)) and
- to propagate renaming of an attribute definition to related attributes (cf., Figure 4.31 (2)).

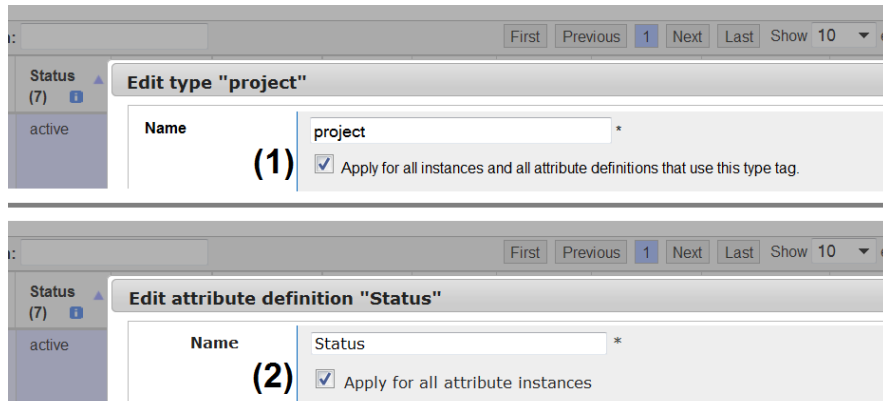


Figure 4.31.: Applying changes in the schema to the instances.

These techniques are implemented by using Tricia batch jobs. This is due to the fact that potentially a huge set of content objects needs to be updated. This also means that such a job possibly needs some time to be executed. Therefore, this may lead to situations where structures manually entered by users are overwritten due to the delayed execution of a batch job. This could be improved by using write locks for entities affected by update jobs until their execution is finished. But the Tricia framework currently does not support such locking mechanisms. However, since all changes to the structure are available in the version history users are aware of unintended modifications and delayed write access is rather less critical.

Using constraints for data consolidation is not possible up to now (cf., Section 3.3.3.1). The implementation of Hybrid Wikis also does not support data-based consolidation as discussed in Section 3.3.4. In an earlier version of the implementation, changes in the data were automatically propagated to the schema. For instance, assigning an attribute to a typed content object automatically led to the creation of an attribute definition in the type(s). However, we observed that users are more inhibited in contributing or changing information structures on the data level if this immediately impacts the schema. Even if this was an observation only, that is, not confirmed by any measurements or controlled user studies, we consciously decided not to propagate changes in the data immediately to the schema. This is also due to the fact that data and schema are modified by persons having different roles, that is, authors rather modify the data whereas tailors tend to change the schema preferably (cf., Section 3.2.1). However, in a later version of Hybrid Wikis it is conceivable that users can decide whether to apply changes in data structures to the schema or not. This would bring authors and tailors more closer together.

4.2.6. Export and import of structured content

Structured information of Hybrid Wikis can be accessed and provided by using the standardized EMF exchange format. Types, attribute definitions, and constraints can be imported

and exported by means of Ecore elements²⁴ (e.g., classes, references, data types), data on the instance level (i.e., wiki pages with type tags and attributes) as XMI files²⁵.

Spreadsheets are frequently used in enterprises since they are lightweight means to create and manage semi-structured information. Therefore, structured content (i.e., type tags and attributes of wiki pages) can additionally be imported from and exported to Microsoft Excel spreadsheets²⁶. In contrast to EMF, the implementation of Hybrid Wikis only allows to exchange structured information of wiki pages, that is, schema information (i.e., types, attribute definitions, constraints) cannot be exchanged based on spreadsheets currently. Importing spreadsheets as structured pages has the several advantages, for example collaborative authoring at a central place, version control, link management, and different tabular views on the same data sets. By exporting structured pages to spreadsheets users can benefit from advanced functional capabilities provided by for example Excel, such as calculations or pivot charts. Structured information can be exported for example in all custom embedded table views (cf., Figure 3.36 (6)) or in the type table view (cf., Figure 3.32 (4)).

We consciously decided to explicitly disable the validation mechanisms provided by Hybrid Wikis (cf., Section 4.2.1.6) when importing data from external sources (e.g., XMI files or spreadsheet). This means data can be imported without being restrained by hard integrity constraints, even in case of strict constraints or rigid types. Only if the Tricia framework detects invalid build-in features (e.g., ambiguous wiki page names) the import is aborted. Any further conflicts with the schema can be fixed after the import. Users are supported in finding violations (cf., Section 3.2.4.1) and by harmonizing invalid values by means of consolidation techniques (cf., Section 3.3.3).

4.2.7. Collaborative information management on federated data sources

In [RW11] the authors are facing the problem that information is scattered across a multitude of special-purpose systems within an enterprise [MCS09]. In particular, they examine how Hybrid Wikis can support federated data integration [Ko01, SL90] scenarios.

The introduced solution integrates information from the enterprise applications Microsoft SharePoint 2010²⁷ (e.g., People and Documents) by using the OData protocol²⁸ and Microsoft Exchange 2010²⁹ (e.g., Contacts and Emails) by means of the *Exchange Web Service* (EWS)³⁰.

The use cases primarily supported by the extended implementation of Hybrid Wikis are:

- View external objects (e.g., Contacts and People) with their structures (i.e., with type and attributes).

²⁴<http://download.eclipse.org/modeling/emf/emf/javadoc/2.7.0/org/eclipse/emf/ecore/package-summary.html#details>; visited on April 05th 2012.

²⁵<http://download.eclipse.org/modeling/emf/emf/javadoc/2.4.3/org/eclipse/emf/ecore/xmi/package-summary.html>; visited on April 05th 2012.

²⁶<http://office.microsoft.com/en-us/excel>; visited on April 05th 2012.


²⁷<http://sharepoint.microsoft.com>; visited on March 10th 2012.


²⁸<http://www.odata.org>; visited on March 10th 2012.

²⁹<http://microsoft.com/exchange>; visited on March 10th 2012.

³⁰<http://archive.msdn.microsoft.com/ewsjavaapi>; visited on March 10th 2012.

- Define queries for external objects according to their structures.
- Specify links to external objects.
- Edit external objects' attributes and synchronize the changes accordingly.
- Change types of external objects locally.
- Add additional attributes to external objects. If the underlying protocol allows the definition of new attributes, synchronize them, if not, store them locally.
- Add additional unstructured content to external objects.

 **Enabling Collaborative Information Management on Federated Data Sources - Analysis, Design and Prototypical Implementation for MS-Sharepoint and MS-Exchange**

Link to source	http://sharepoint2010:1234/Lists/Lectures/DispForm.aspx?ID=1
External Data Source	 Lectures
Last time reloaded	1 minute ago

Additional Content (2)
 Information ist in der heutigen Gesellschaft und speziell im Umfeld IT-gestützter Unternehmen ein bedeutendes Gut, demzufolge ist dessen Management eine wichtige Aufgabe. Viele Unternehmen verdienen ihr Geld damit, Systeme zu entwickeln und zu verkaufen, welche betriebliches Informationsmanagement ermöglichen, z.B. Enterprise Content Management Systeme, Datenbanksysteme oder einfache Dateiablagen.
 Nicht selten findet man also in Unternehmen eine ganze Landschaft an Systemen vor, welche verschiedenste Arten von Information verwalten, was vorwiegend dazu führt, dass es für Benutzer schwieriger wird, eine bestimmte Information im Unternehmen zu finden. Eine Möglichkeit, mit diesem Problem umzugehen, ist es, ein System zu bestimmen bzw. einzuführen, in welchem verschiedene Informationsquellen integriert werden und wodurch dieses zentrale System eine einheitliche Oberfläche für den Einstieg in den Informationsdschungel des Unternehmens bietet.
 Diesem Ansatz der Datenintegration widmet sich diese Bachelorarbeit, wobei neben einer theoretischen Sicht auf die Probleme und Herausforderungen der Datenintegration zugleich






Types: lecture (1)	
Title	Enabling Collaborative Information Management on Federated Data Sources - Analysis, Design and Prototypical Implementation for MS-Sharepoint and MS-Exchange
Speaker	(3)  Thomas Georg Reschenhofer, Bernhard Waitl
StartTime	 26 Jun 2011 22:00:00 GMT
Betreuer	(4)  Neubert Christian (5)  Buechner Thomas
Professor	 Matthes Florian

Figure 4.32.: An external object (from SharePoint) describing a lecture with unstructured, additional content, external attributes (e.g., Speaker), and locally stored attributes (e.g., Betreuer) which are linking to external objects (from Exchange) representing persons (e.g., Neubert Christian) according to [RW11].

Figure 4.32 depicts an external object (originating from a SharePoint server) describing a lecture (cf., Figure 4.32 (1)) with unstructured, additional content (cf., Figure 4.32 (2)), external attributes (e.g., Speaker, cf., Figure 4.32 (3)), and locally stored attributes (e.g., Betreuer, cf., Figure 4.32 (4)) which are linking to external objects (originating from an Exchange server) representing persons (e.g., Neubert Christian, cf., Figure 4.32 (5)).

The solution additionally allows to configure the content types (e.g., Lecture, Person) which are to be integrated from the external datasources (e.g., SharePoint, Exchange). Type tags are used to indicate the content types and attributes to represent the external objects' properties. The mapping between type tag and content type (e.g., for SharePoint Lectures the type tag lecture is used) as well as the set of attributes that is shown when displaying an external object (e.g., Title, Speaker, StartTime) can be configured.

In this way, external information is smoothly integrated with the local structures. For instance, it is possible that objects shown in the type table view are either external objects or local

wiki pages if both using the same type tag. Furthermore, queries can be specified uniform across all structured information, that is, uniform across external and internal objects since both using the concepts provided by Hybrid Wikis.

The results of that thesis indicate that Hybrid Wikis support federated data integration scenarios. However, the concepts provided by Hybrid Wikis are limited with regards to their expressivity. That means in some cases external information cannot be represented in Hybrid Wikis that precise as in the source application (e.g., if the external data source provides advanced data types, such as currency). Nevertheless, the authors show that Hybrid Wikis provide a holistic view on information within an enterprise scattered across different application repositories. It is possible to integrate information from both kinds of data sources, structured and unstructured. Therefore, by connecting information islands within enterprises Hybrid Wikis help to bridge the gap between structured and unstructured information which is one of the key challenges in knowledge management research [MN11a].

In this chapter, we present approaches related to Hybrid Wikis. First, we briefly describe findings from our survey on integrated Enterprise 2.0 applications. Subsequently, we compare Hybrid Wikis with tools and prototypes supporting information structuring described in scientific literature.

In one of our research projects, called Enterprise 2.0 Tool Survey [BMN09, BMN10b], we analyzed the functional capabilities of open source and commercial Enterprise 2.0 platforms. In the first iteration, conducted in 2008, we examined seven prominent social software solutions selected according to [Dr07], namely Alfresco Labs¹, Atlassian Confluence², Groupswim³, Jive Clearspace⁴, Liferay Social Office⁵, Microsoft Office SharePoint Server 2007⁶, and Socialtext⁷. Additionally, we included our Enterprise 2.0 platform Tricia.

We identified the core content objects underlying each platform (e.g., wiki pages) by creating conceptual class diagrams using reverse engineering techniques and derived 51 Enterprise 2.0 services (e.g., tag support for all content objects) orthogonally applicable to the content objects. Finally, we grouped these services in 13 more general categories (e.g., tagging), resulting in a multi-dimensional classification and evaluation framework for integrated Enterprise 2.0 applications. Based on this framework, we evaluated to which degree a tool fulfills individual services by applying ratings between 0 and 4 (i.e., 0 stands for no capabilities, 4 for complete coverage of the service). In [NST09a, NST09b] we put our framework in context with similar approaches for classifying Enterprise 2.0 platforms.

¹<http://www.alfresco.com/products/collaboration>; visited on February 22nd 2012.

²<http://www.atlassian.com/software/confluence>; visited on February 22nd 2012.

³The Groupswim product was acquired by salesforce.com in the meantime (cf., <http://www.cloudave.com/1072/breaking-salesforce-com-buys-groupswim>).

⁴<http://www.jivesoftware.com>; visited on February 22nd 2012.

⁵http://www.liferay.com/web/guest/products/social_office; visited on February 22nd 2012.

⁶<http://www.microsoft.com/Sharepoint/default.aspx>; visited on February 22nd 2012.

⁷<http://socialtext.com>; visited on February 22nd 2012.

5. Related Work

The results⁸ from 2008 show that none of the examined approaches provides information structuring capabilities⁹. Only MS SharePoint 2007 allows to change some meta data of files, in particular the title property of a file.

In [Mi10] the author extends the framework from 2008 with additional services and categories based on the analysis of a newer version of the MS SharePoint server. In particular, the author shows the evolution of the Enterprise 2.0 services and categories from 2007 to 2010 by examining the functional capabilities of the two platforms Microsoft SharePoint Server 2010 and Tricia. Based on this work, we additionally evaluated the capabilities of XWiki¹⁰, TWiki¹¹, and a newer version of Jive and updated the framework according to our findings.

Survey result matrix 2010

Category	Service	Vendors				
		Jive	Microsoft SharePoint Server 2010	TWiki	Tricia	XWiki
Access Control	Uniform flexible and finegranular access control concept for all content types	●	●	●	●	●
	Functional groups for access control	●	●	●	●	●
	Smooth transition between the usage modes not logged on and logged on	●	●	●	●	●
	Spam avoidance	◐	○	◐	◑	◐
	OpenID support	○	○	◐	●	◐
	Content of any type may be made available for anonymous users	●	●	●	●	●
	Creation of groups and invitation of new members by users	●	◐	●	●	◐
Authoring	Autosave	●	○	○	●	○
	Concurrent editing	●	◐	●	●	●
	In place editing	◐	○	○	●	○
	Structuring of content	○	◐	◐	◐	◐
	Input support for link creation	◐	◐	◐	●	●
	Offline editing	○	○	○	○	◐
	Templates for structured content	○	●	●	●	●

Figure 5.1.: sebis Enterprise 2.0 Tool Survey result matrix 2010 showing some categories, services, and vendor ratings.

The findings¹² from 2010 (cf., Figure 5.1) show that some vendors already include basic functions to support the structuring of content at this time. However, in SharePoint only predefined types of attributes can be assigned by adding additional columns to lists and libraries, XWiki enables users to structure information by programming macros. The fact that leading social software vendors include structuring capabilities for content indicates an increasing demand in enterprises. But the results also show that information structuring is not yet well supported.

Based on these findings, in [Hu11] we carried out a literature review taking into account web-

⁸<http://www.matt.hes.in.tum.de/wikis/enterprise-2-0-survey/2008-home>; visited on February 22nd 2012.

⁹In 2008, Tricia provided a traditional wiki only, that is, a wiki without structuring support.

¹⁰<http://www.xwiki.org>; visited on February 22nd 2012.

¹¹<http://twiki.org>; visited on February 22nd 2012.

¹²<http://www.matt.hes.in.tum.de/wikis/enterprise-2-0-tool-survey-2010/home>; visited on February 22nd 2012.

based tools and prototypes supporting information structuring. In a first iteration, the author identified 19 relevant approaches described in 31 scientific sources, including Hybrid Wikis. Subsequently, the main characteristics (e.g., data model, input support, views) are extracted from these sources and summarized in a multi-dimensional catalog. The catalog also indicates the relation between literature source and characteristic, that is, which characteristic is mentioned in which source. Additionally, the characteristics are grouped in three more general categories, namely system of structure, input & presentation, and navigation & search. Based on this catalog, the author compared the identified characteristics with Hybrid Wikis. In particular, he identified characteristics only available in Hybrid Wikis (e.g., import and export of structured content via EMF), not existing in Hybrid Wikis (e.g., inheritance of types), and partially existing in Hybrid Wikis (e.g., widgets facilitating data input, such as date picker). His work concludes with a discussion about how characteristics not or only partially existing could be beneficially integrated in Hybrid Wikis. For example, he proposes to support inheritance of types to avoid the assignment of multiple type tags.

[Hu11] focuses on functional aspects of tools and prototypes supporting information structuring discussed in scientific literature. Based on this work, we conducted an additional literature analysis in order to broaden the scope and to compare Hybrid Wikis with other approaches on a more conceptual level. The remainder of this chapter presents the results of this analysis.

In the period 1st February 2012 - 29th February 2012, we accessed the databases CiteSeerX¹³, Google Scholar¹⁴, IEEE Xplore Digital Library¹⁵, ISI Web of Knowledge¹⁶, SpringerLink¹⁷, Microsoft Academic Search¹⁸, and the digital libraries of Technische Universität München. We focused our study on collaborative information structuring in enterprise web environments by applying the keywords *structure*, *structured*, *structuring*, *data*, *information*, *content*, *wiki*, *emergent*, *adaptive*, *enterprise*, *web*, *collaboration*, *collaborative* in different combinations.

We identified 28 approaches (including 17 semantic wikis) to be relevant for this thesis. The selection was guided primarily taking into account literature

- elaborating on web-based tools or prototypes (or parts thereof) that
- support or facilitate information structuring in
- a collaborative or personal web environment.

In the next sections, we present the identified approaches using the following structure:

- Briefly introduce the core idea and key concepts underlying an approach,
- compare it with Hybrid Wikis by highlighting the main differences and commonalities, and
- discuss pros and cons.

¹³<http://citeseerx.ist.psu.edu>; visited on February 29th 2012.

¹⁴<http://scholar.google.com>; visited on February 29th 2012.

¹⁵<http://ieeexplore.ieee.org>; visited on February 29th 2012.

¹⁶<http://wokinfo.com>; visited on February 29th 2012.

¹⁷<http://academic.research.microsoft.com>; visited on February 29th 2012.

¹⁸<http://academic.research.microsoft.com>; visited on February 29th 2012.

We also included the W3C standard Semantic Web¹⁹ in our results (cf., Section 5.1). Even if this standard is not directly realized as a tool or prototype its ideas and concepts are the foundation for several of the other approaches. All approaches are described and discussed to the best of the authors knowledge based on the sources cited in each section.

5.1. Semantic Web

The Semantic Web is a W3C standard to represent data in the Internet, often referred to as Linked Data [BHBL09] or Open Data [Au08]. The W3C terms it as “a common framework that allows data to be shared and reused across application, enterprise, and community boundaries”. The inventor of the web, Tim Berners-Lee, defines it as “a web of data that can be processed directly and indirectly by machines” [DSW06], furthermore he describes it as “a web of data, in some ways like a global database” [BL98].

This means the Semantic Web is an approach to give meaning to web resources represented as unstructured hypertext using meta data in order to make them processable outside the environment they were created [KC04].

In particular, the approach aims at:

- Link, reuse, and search data across different information (re)sources in the World Wide Web.
- Exploit new data connections using inference techniques [Ha04].

This meta data is represented by concepts provided by RDF [KC04]. The RDF core concept is a Graph Data Model representing a resource’s meta data as triples, each consisting of a subject, a predicate, and an object [Ha04]. A resource identified by a *Uniform Resource Identifier* (URI) represents the subject. It provides a set of named predicates, its properties. A predicate can also be a (named) resource. The property’s value represents the object. A value can either be a literal or a resource again. An RDF graph consists of nodes represented by the subjects and objects. The graph is directed, this means “it always points toward the object” [Ha04] (cf., Figure 5.2).

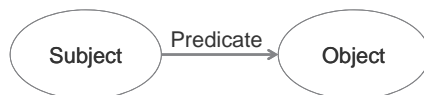


Figure 5.2.: An RDF triple represented by subject, predicate, and object according to [Ha04].

The RDF Vocabulary Definition Language, also called *RDF Schema* (RDFS), aims at the definition of a common vocabulary describing a set of RDF resources within a specific application domain. Similar to RDF it is recommended by the W3C [BG04]. The standardized vocabulary allows the definition of classes and properties. For instance, a resource can be declared as a class. Then it can be expressed that another resource is an instance of that class using a property.

¹⁹<http://www.w3.org/2001/sw>; visited on February 2nd 2012.

Due to the fact that RDFS is limited with regards to its expressive power, it is extended by the *Web Ontology Language* (OWL). In particular, “OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. ‘exactly one’), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes” [MH04]. OWL provides three increasingly expressive sublanguages, namely OWL Lite, OWL DL, and OWL Full. They are designed to support specific communities of developers and users [MH04]. For instance, OWL lite enables users to build a classification hierarchy and simple constraints, OWL DL focuses on computational completeness and decidability, and OWL Full supports the maximum expressiveness of RDF [MH04].

In order to access RDF structures the *SPARQL Protocol and RDF Query Language* (SPARQL) is utilized. Similar to SQL this language allows to retrieve and manipulate RDF data. According to [PS08], “SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions”. Query results are represented as an RDF graph. The complete technology stack provided by the Semantic Web initiative is depicted in Figure 5.3.

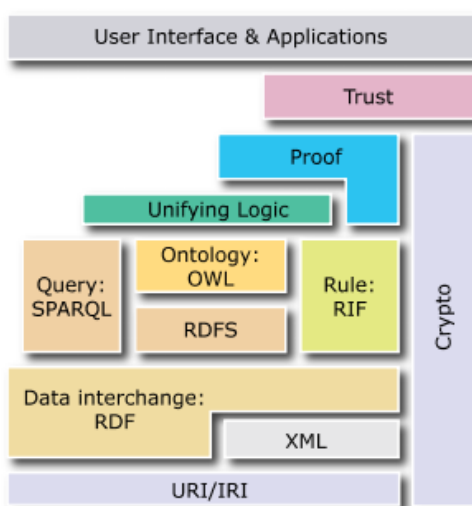


Figure 5.3.: Layers of the Semantic Web. Source: <http://www.w3.org/2007/03/layerCake-small.png>; visited on February 10th 2012.

The Semantic Web approach provides means to represent the concepts of Hybrid Wikis. On the instance level, a content object with URL, attributes, and values represents an RDF triple. Types, attribute definitions, and constraints from the schema can be mapped to elements of RDFS and OWL. Furthermore, the design goals underlying RDF are quite similar to those of Hybrid Wikis. Some of these goal are [KC04]:

- “Having a simple data model”.
- Using an extensible vocabulary.
- “Allowing anyone to make statements about any resource”.

However, the fact that OWL exists in three variants differing with regard to the expressive-

5. Related Work

ness shows that in some cases it matches not exactly the application domain. We developed Hybrid Wikis starting with a minimal set of structuring concepts which we incrementally extended according to business users' needs necessary for their day-to-day work. We consciously decided to limit Hybrid Wikis regarding the expressiveness in querying data by allowing to combine simple conjunctive search filters only and avoiding the definition of join operations. In contrast, the Semantic Web approach supports complex inference and reasoning rules [Ha04]. Furthermore, RDF is a standard intended to share resources across application, enterprise, and community boundaries. Hybrid Wikis are primarily designed for the application within enterprises. That means, in most cases it is not necessary to expose the content objects' structure globally in the web. However, in some cases it can be beneficial to make structures available for external stakeholders, for example in order to share (structured) product descriptions with customers. Even if Hybrid Wikis currently do not support the RDF standard it would be simple to map the concepts to RDF and OWL. Then RDF/OWL statements could additionally be provided as annotated meta data within the page content (hidden from the users) in order to share structures globally in the web (e.g., to make the content of Hybrid Wikis accessible to semantic web search engines [GMM03]). Even if Hybrid Wikis are not based on Semantic Web technologies other wiki approaches follow this idea (cf., Section 5.5).

5.2. Wiki templates

Wiki templates, such as those heavily used in the Wikipedia project (cf., Figure 5.4), enable authors to reuse content structures among wiki pages.

```
1  {{Infobox Town AT |
2    name = Innsbruck |
3    image_coa = InnsbruckWappen.png |
4    image_map = Karte-tirol-I.png |
5    state = [[Tyrol]] |
6    regbzkg = [[Statutory city]] |
7    population = 117,342 |
8    population_as_of = 2006 |
9    pop_dens = 1,119 |
10   area = 104.91 |
11   elevation = 574 |
12   lat_deg = 47 |
13   lat_min = 16 |
14   lat_hem = N |
15   lon_deg = 11 |
16   lon_min = 23 |
17   lon_hem = E |
18   postal_code = 6010-6080 |
19   area_code = 0512 |
20   licence = I |
21   mayor = Hilde Zach |
22   website = [http://innsbruck.at] |
23 }}
```



Innsbruck	
	
Country	Austria
State	Tyrol
Administrative region	Statutory city
Population	117,342 (2006)
Area	104.91 km ²
Population density	1,119 /km ²
Elevation	574 m
Coordinates	47°16′N 11°23′E﻿ / ﻿47.267°N 11.383°E﻿ / 47.267; 11.383
Postal code	6010-6080
Area code	0512
Licence plate code	I
Mayor	Hilde Zach
Website	www.innsbruck.at

Figure 5.4.: Wikipedia template representing Austrian towns applied to Innsbruck according to [AL07].

In [HLS05], the authors discuss the need for structure in wikis by introducing the following five requirements:

- users want to create structured content,
- content should be readable for authors (i.e., should not contain any layout elements),
- users should be aware of unintended changes to the structure,
- users should be able to change the structure according their needs, and
- user-created structure should be reusable for similar tasks.

The presented solution to these requirements are wiki templates. Users can assign a template to a wiki page when a new page is created. A template has a name and consists of two further parts both defined in an extended wiki markup language. The edit part contains named placeholder fields that can be filled when a page using this template is edited (cf., Figure 5.5). The display part defines the appearance of a page for wiki readers. Therefore, a template describes both the page's structure as well as its layout.

Although Hybrid Wikis explicitly do not support templates, attribute and type (-tag) suggestions based on a statistical analysis of type tag and attribute combinations enable authors to reuse well established (type) structures, similar to templates. However, Hybrid Wikis differ from wiki templates as follows:

- Assigning a template to a page can be considered as assigning a type. Therefore, attributes can only be used together with a type, assigning attributes independently is not possible.
- It is only possible to assign one template to a wiki page.
- From the author's knowledge it is not clear if a wiki template can be removed from a page or replaced by a different template without losing the filled values.
- New attributes can only be specified by changing the template which requires the usage of a specific template editor. This means users need to change the current edit context (i.e., from page edit to template edit).
- Users can tailor templates only by using a wiki markup-like definition language, also including HTML-like elements. Therefore, customizing a template requires web programming experience.
- Templates describe the structure and the appearance of wiki pages. In Hybrid Wikis structure and content are separated and three different built-in views are provided in order to represent pages according to their degree of structure.
- The approach enables users to define who is allowed to modify a template within a wiki. Hybrid Wikis follow the wiki way [LC01] since all wiki editors are allowed to modify types, attribute definitions, and constraints.
- When a page is edited the placeholders can only be filled with textual content, that is, no further data types are provided (e.g., number and date).

As indicated, quite a lot differences between both approaches exists. However, Hybrid Wikis meet the requirements as introduced before. In Hybrid Wikis

- users can create structured content by using types, attributes, and constraints,

5. Related Work

- content is readable for authors since built-in content and structure are separated,
- tailors are aware of unintended changes since types and attribute definitions can be observed, are under version control (cf., Figure 3.2.4), and validation messages indicate derivations from defined structures,
- users can change the structure according their needs on the instance level (wiki pages) or directly in the schema, and
- user-created structure can be reused by means of type assignments and attribute suggestions, even across space boundaries.

```
<box>
!!Bibliographic Data
|Author: | <wikiTextInput id="author" />|
|Title: | <wikiTextInput id="/pageTitle" />|
|In: | <wikiTextInput id="volume" />|
|Place: | <wikiTextInput id="place" />|
|Year: | <wikiTextInput id="year" />|
|Pages: | <wikiTextInput id="pages" />|
|URL: | <wikiTextInput id="url" />|

!!Summary
|Research Questions: | <wikiTextInput lines="8"
                        id="research Questions" />|
|Research Methods: | <wikiTextInput lines="8"
                        id="research Methods" />|
|Results: | <wikiTextInput lines="8"
                        id="research Results" />|

<submitButton /><cancelButton />
</box>
```

Figure 5.5.: Wiki edit template specifying two sections by using wiki markup (Bibliographic Data and Summary) with properties (e.g., Author) and placeholder input fields (wikiTextInput) according to [HLS05]

In [DIVZ08] different approaches using wiki templates are compared to each other. In a first step, the authors identified user activities that can benefit from wiki templates taking into account the roles visitor, editor, and tailor. Subsequently, two different templating models are identified, namely functional templating and creational templating. In the former case, templates are invoked by name and parameters, such as used in [Pa09]. This has the advantage that changes in the template are automatically available in the instance pages, that is, templates and instances never diverge. A disadvantage is that the template is not directly editable in the page instance since the template's source is maintained in a separate page. In case of creational templating, a template page is simply copied for purpose of reuse. Therefore, the template source is always directly available in the instance page. However, this has the disadvantage that changes in the instance are not reflected in the template (and vice versa) since they are not connected to each other.

Based on one of the authors earlier works [DIZ06], the notion of Lightly Constrained Templating is introduced which allows to combine creational templating with integrity constraints. In this approach a validation rule is created when a template is copied. This rule checks if the

structure of the template conforms to the instance's structure when the instance page is saved. User are made aware of rule violations by showing validation messages (cf., Figure 5.6). This counteracts the deviations of templates and instances.

Hybrid Wikis tend to follow the functional templating approach since a type assignment results in suggesting attributes related to that type. As mentioned above, this approach has a disadvantage since the structure (i.e., template) is defined in a different place than the content, leading to non-linearity of the markup [DIVZ08]. However, in Hybrid Wikis attributes and types can freely assigned to pages without the need to change the context. Only in case of defining constraints or binding an attribute to a type, the user needs to leave the current context. In a future version of Hybrid Wikis, it is conceivable that constraints can be defined and modified directly on the page level.

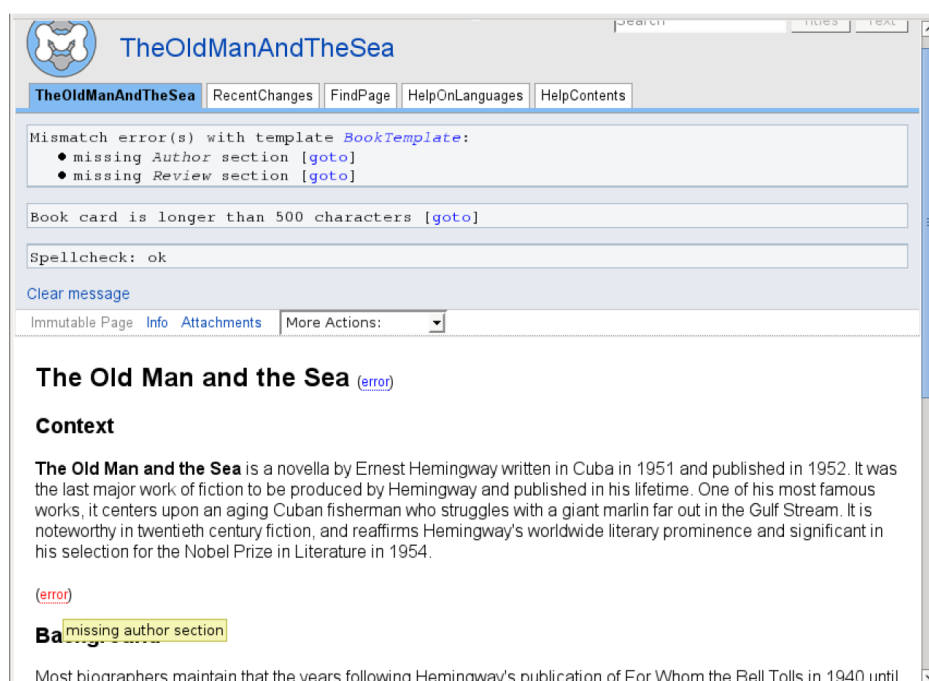


Figure 5.6.: A wiki page using Lightly Constrained Templating mechanisms indicating validation errors caused by deviations from the master template according to [DIVZ08].

Templates provide means to automatically extract structured information from wikis, the template name can be interpreted as a type, named placeholders as attribute keys, and link values filled in placeholder fields on instance pages as references between types. Approaches following this idea in the context of Wikipedia are for example [AL07, IB07, WW08] or the DBpedia project (cf., Section 5.4).

5.3. DynaTable

In [Ar09] DynaTable is introduced, an extension for the MediaWiki engine supporting structured data. The authors address the problem that tables directly defined in the page con-

5. Related Work

tent are often copied in order to include them in other pages for specific information needs (e.g., showing only a subset of the original table’s columns in a different order). The solution provided allows to

- maintain named data tables and table data separately and in a dedicated space (i.e., not in the content of a regular page),
- include tables in page content using a specific markup tag referencing their names (cf., Figure 5.7), and
- create custom table views (e.g., by selecting a subset of columns).

The approach is similar to the custom embedded table view as introduced in Section 3.4.2. The difference is that in DynaTable data cannot be entered at the place the table is displayed. That means users have to leave the current context to enter data in a dedicated space. In Hybrid Wikis data can directly be entered in the table view according to the current search filter resulting in newly created structured wiki pages.

Company	Logos	Founded	Location	Revenue
Intel	Image:Intel logo.jpg	USA		
SD Squared		January 2009		\$27.42
Apple	Image:Apple logo.png	April 2005		\$7.42
Microsoft		1492	Seattle	\$8

Figure 5.7.: Code embedded in a wiki page (lower part of the figure) to render a table (upper part of the figure), named Companies, with data defined in a separate space according to [Ar09].

5.4. DBpedia

In [Au08, Ko09, Le09] DBpedia is introduced as a community-driven project with the objective to extract structured information from Wikipedia content and to provide this information publicly in the web. This way, Wikipedia data can be referenced from other web resources and queried similar to records in a database. This makes it possible to build applications and websites based on Wikipedia datasets. By interlinking with other open data collections (cf., Figure 5.8) DBpedia serves “as a nucleus for an emerging Web of open data” [Au08].

MediaWiki, the platform underlying Wikipedia, supports simple wiki markup and wiki templates in order to present content in a specific way (cf., Section 5.2). Infoboxes are special kinds of templates describing wiki pages with similar content, such as cities (cf., Figure 5.4). Templates are represented as wiki markup and embedded in the wiki page content. The DBpedia approach syntactically analyses the markup of Wikipedia pages using pattern matching techniques in order to extract RDF statements from for example templates and links. More details about the extraction algorithm are described in [AL07].

Even if the attribute box in Hybrid Wikis is inspired by the layout of Wikipedia’s infoboxes²⁰,

²⁰<http://en.wikipedia.org/w/index.php?title=Help:Infobox&oldid=479756931>; visited on March 15th 2012.

our approach does not support the idea of embedding templates in the content. Structured and unstructured information is clearly separated and the structure is only presented similar to infoboxes. Furthermore, in the DBpedia project structured information is automatically extracted from the wiki contents. In contrast, in Hybrid Wikis information is manually structured and users are guided by automatically generated suggestions. However, it would be interesting to evaluate which parts of the built-in content in Hybrid Wikis can be utilized in order to additionally facilitate information structuring. Since DBpedia is part of the Semantic Web initiative a primary goal is to make structure publicly available in the web. In a future version, Hybrid Wikis could expose their structures as an RDF graph publicly in the web. By doing so, Hybrid Wikis could also be a part of the interconnected graph of open datasets as proposed by the DBpedia approach.

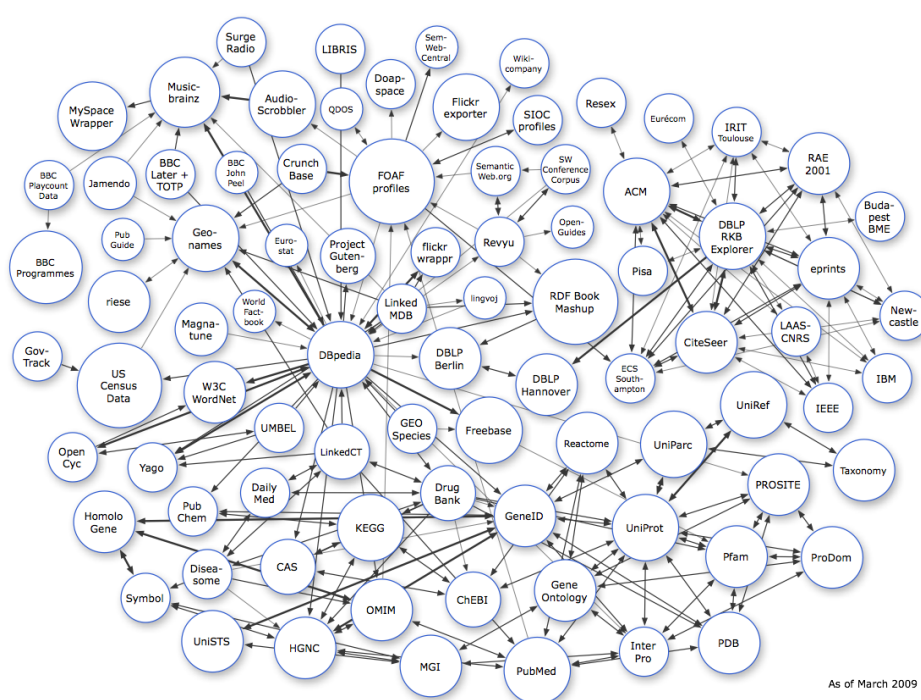


Figure 5.8.: DBpedia as a hub of interlinked data according to [Bi09].

5.5. Semantic wikis

Semantic wikis [Bo09, CCT04, KSV07, Sc08] enhance traditional wikis, such as the MediaWiki [Ba09], with capabilities to access the pages' content similar to a database. The content is additionally enriched with meta data represented in a formal language. This language is interpretable and processable by machines. In most of the existing semantic wikis meta data is represented by concepts provided by the RDF standard. By means of RDF statements it is possible to annotate elements within unstructured wiki pages, such as text fragments and links. For instance, it can be expressed that an individual page represents a concept (e.g., *project*) using OWL annotations and another page being an instance of that concept, additionally providing some attributes (e.g., *budget*) using RDF properties. The page with its URL represents

the subject, the used annotation the predicate, and the annotated element the object. These three constituents form an RDF triple (cf., Section 5.1). For purpose of querying data SPARQL is used. However, some semantic wikis also use their own representation of annotations and query languages.

Semantic wikis are applied in several domains for different purposes. For instance, in [HL09] it is evaluated how they bring benefits in the area of requirements engineering, in [TGP11] it is explained how they support project management activities, in [Ga10a] how quality management in software development projects is facilitated, in [Or06b] requirements for personal knowledge management are addressed, and in [DH10] wiki-based modeling of processes and UML (class-)diagrams is presented. The capabilities of different semantic wiki approaches are compared in several studies (e.g., [Bu11a, De05a, HL09, Mi08, Sc08]).

In the following, we put the most prominent approaches in the context of Hybrid Wikis and highlight the main differences and commonalities.

5.5.1. Semantic MediaWiki

The Semantic MediaWiki project²¹ [KVV06] is the most prominent example in the category of semantic wikis. This project adds “database-like structuring and querying capabilities on top of an existing wiki, without requiring users to develop or adhere to a rigid database schema when authoring content”. Additionally, it “includes various features for browsing, searching, and aggregating the wiki’s content”²² and allows users to embed queries in the pages. Even if the annotation language is given in wiki syntax, meta data can be exported to the standardized format RDF/OWL. In [Vö06] the core concepts of Semantic MediaWiki are introduced:

- Categories, to classify pages.
- Typed links, to classify links.
- Attributes, to represent simple properties of a page.

In that article, the authors propose to use these concepts to make Wikipedia articles accessible to external applications. Categories corresponds to types, typed links [KVV05] to link values, and attributes to key-value pairs (not having link as data type) in Hybrid Wikis.

Unlike in Hybrid Wikis, in Semantic MediaWiki this meta data can only be added to the pages by directly editing the markup in wiki syntax. For instance, in order to express that the page describing *Development Project A* is of type *project* and has a specific *budget* the user needs to manually produce the markup as depicted in Figure 5.9.

In this example, only the value *200.000* is shown in the content when the page is displayed, the attribute key *has Budget* is hidden. Additionally, it is indicated that the page is categorized as a *Project* when viewing the page. The fact that users need to learn and use a special annotation syntax when editing a page makes it laborious for them to manage structured and unstructured content. Additionally, users are not assisted when entering attributes and categories. This means they need to know existing structures in order to (re)use them. This is due to the

²¹<http://semantic-mediawiki.org>; visited on February 10th 2012.

²²http://semantic-mediawiki.org/wiki/Help:Browsing_and_searching; ; visited on May 10th 2012.



Figure 5.9.: Semantic annotations used in the markup editor provided by Semantic MediaWiki.

fact that neither attribute nor type suggestions are provided built-in. Potentially, this leads to redundancies and inconsistencies in the usage of vocabulary and terms. Furthermore, the page’s structure is only available in a special view in Semantic MediaWiki. Instead, in Hybrid Wikis the structures (types and key-value pairs) are treated as ‘first class content’ and not as additional meta data hidden in the markup. Therefore, available structures are always visible when viewing a content object per default (cf., Section 3.4).

Similar to Hybrid Wikis the Semantic MediaWiki approach aims at the collaborative development of models. In [Gh08] the approach is used to collaboratively develop an OWL ontology representing a specific application domain in different evolution phases. In a preliminary phase, the knowledge about the domain to be modeled is acquired from domain experts resulting in informal descriptions representing the domain in structured wiki pages. In a subsequent phase, these descriptions are merged together resulting in a formal model, the OWL ontology. The authors point out “that the informal model is not kept up-to-date with changes made in the formal model at this stage because reflecting the changes made into an OWL ontology back to SMW is not a trivial task”. In Hybrid Wikis it is not required to export ontologies for merging purposes and re-import the merged ontology afterwards since the consolidation techniques as introduced in Section 3.3.3 facilitate the harmonization of different schemes within the application, additionally harmonizing related instances accordingly, if needed.

5.5.2. Semantic Enterprise Wiki (SWM+)

The Semantic Enterprise Wiki (SWM+)²³ provides a set of open source extensions to compensate some shortcomings of the Semantic MediaWiki approach. In [Pf08] interface enhancements to Semantic MediaWiki are introduced and evaluated.

²³http://www.smwplus.com/index.php/Semantic_MediaWiki_Plus; visited on February 10th 2012.

In particular, this article addresses the research questions:

- “How can users unfamiliar with semantic technologies be helped and motivated to specify the many necessary typed links in a way that results in a useful web of annotations?”
- “And how can visitors of the resulting pages be helped to make maximal use of the added semantic information?”

In this work, the authors introduce a separate graphical annotation mode, called semantic toolbar. This toolbar helps to provide semantic meta data without the need to manually annotate contents within the wiki markup, additionally supported by autocompletion. When activating the toolbar it shows the page’s categories and properties (cf., Section 5.5.1) as a list (cf., Figure 5.10). Furthermore, it provides attribute suggestions as well as means to specify queries and to browse the ontology underlying the meta data. In [HSP09] these extensions to Semantic Media Wiki are further detailed.

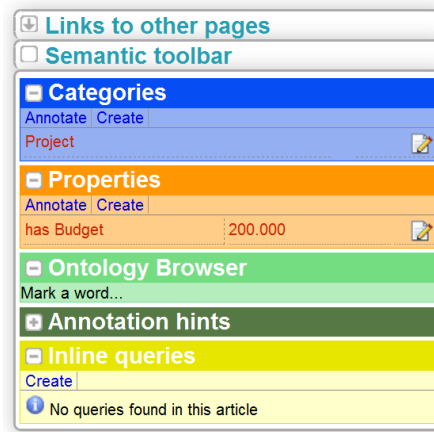


Figure 5.10.: Semantic Toolbar provided by SWM+ showing categories and properties.

The approach differs from Hybrid Wikis as follows:

- A separate annotation mode (i.e., the semantic toolbar) is needed to enter structured information. In Hybrid Wikis structures are shown as part of the wiki page and can directly be edited in-place.
- Users have to create and edit both, the content of the wiki page and the semantic annotations. Additionally, users have to be aware of content and annotations being synchronized, which is a laborious and error-prone task.
- No attribute (property) suggestions based on the types (categories) are provided.

5.5.3. AceWiki

In [Ku08a, Ku08b, Ku08c] AceWiki is introduced. The aim of this project is to create a knowledge base using *Natural Language Processing* (NLP) techniques. In particular, AceWiki uses

the controlled natural language *Attempto Controlled English* (ACE)²⁴. A graphical predictive editor [KS08] helps to create syntactically correct sentences according to the underlying ACE (cf., Figure 5.11), additionally supported by autocompletion. These sentences can automatically be translated into logic and in most cases in an OWL ontology. In [Ku09] case studies showed that AceWiki can serve as a tool to support “domain experts with no background in formal methods” in creating “expressive ontology languages”. Besides AceWiki, many further approaches [HZG09, WG07, ZMG08] combine NLP with wiki technologies.

Similar to Hybrid Wikis NLP-based wikis try to avoid users, in particular domain experts, getting in contact with complex technologies, such as OWL and RDF, in order create a knowledge base. That means, both approaches facilitate the structuring of data. However, in NLP-based wikis this is achieved by using controlled natural languages, in Hybrid Wikis by using a familiar interface based on forms and spreadsheet-like tables.

Development Project DISS

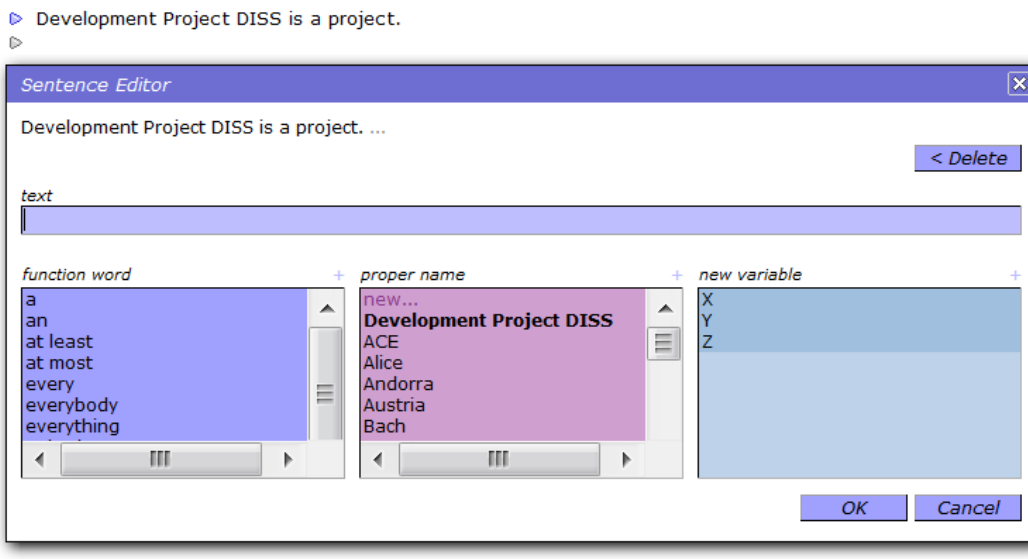


Figure 5.11.: Predictive editor in AceWiki using the controlled natural language ACE to create sentences.

5.5.4. OntoWiki

In [TFH10, DAR06, HBS06] OntoWiki²⁵ is introduced as a tool supporting “agile, distributed knowledge engineering scenarios”. Its implementation is based on Powl [Au05a, Au06], a framework for Semantic Web application development. In [ADR06] the authors motivate their work by identifying obstacles regarding wikis using semantic annotations within the markup, such as in the Semantic MediaWiki project (cf., Section 5.5.1).

In particular, they see obstacles regarding

²⁴<http://attempto.ifi.uzh.ch/site>; visited on February 10th 2012.

²⁵<http://ontowiki.net/Projects/OntoWiki>; visited on February 10th 2012.

5. Related Work

- the usability since editors have more syntactic possibilities and
- the scalability since changes in the knowledge base require to parse the wiki text and apply them accordingly.

The main goal of the approach is “to rapidly simplify the presentation and acquisition of instance data from and for end users”. For this purpose, OntoWiki provides a generic user interface for viewing and editing RDF resources (cf., Figure 5.12). This interface tries to avoid mixing “text editing with knowledge engineering”. This goal is similar to Hybrid Wikis in which unstructured and structured information is clearly separated from each other. But in consequence this also means that users in OntoWiki can access structured data only. That is, it is not possible to provide information without giving them a meaning. Even if a property can have rich text as value (cf., Section 3.1.5.4), it is not possible to create a resource (e.g., wiki page) only consisting of plain unstructured content. Therefore, users are always forced to provide a key, even if this is not required in any case. In contrast, Hybrid Wikis support both, plain unstructured content²⁶ and structured elements (i.e., attributes and types) as well as transitions between them.

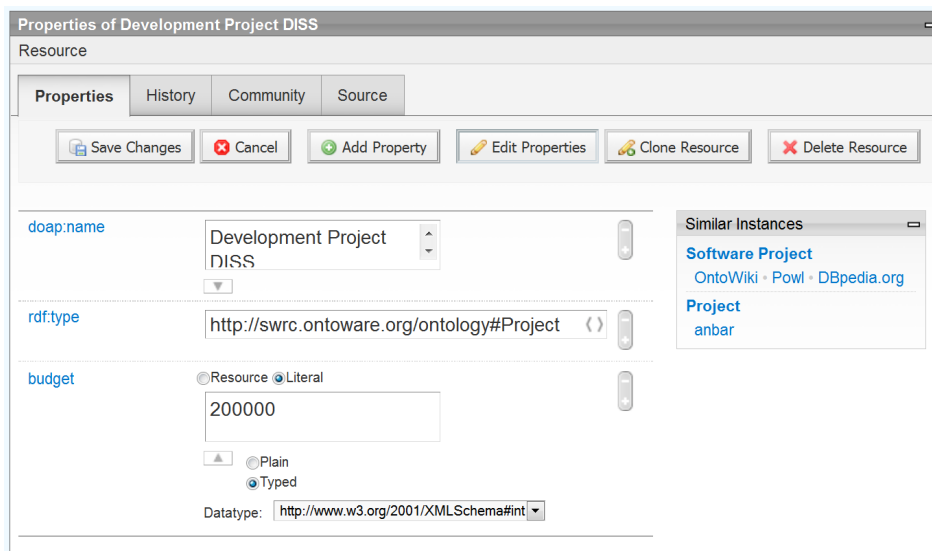


Figure 5.12.: Resource editor provided by OntoWiki.

Furthermore, OntoWiki supports domain specific views, such as maps and calendars, inverse links, suggestions for similar instances (of the same type), and inplace editing facilitating the creation of and navigation in information structures.

²⁶Additionally, unstructured content can contain semi-structured information, such as (untyped) links and tables (cf., Section 3.1.2), but no meta data.

5.5.5. Kaukolu

In [Ki06] Kaukolu²⁷ is introduced as a semantic intranet wiki using RDF annotations to represent information structures. It is part of the Mymory project [Ki08], following the idea of contextualized annotations (e.g., reading annotations gathered by means of an eye tracker).

In [Ki06] Kiesel reveals the following problems with existing semantic wiki solutions.

- Integration of imported RDF data seems to be amendable since ontologies cannot be modified by the wiki's users.
- A subject of an RDF triple inevitably represents the URI of the wiki page the triple is located at. For instance, it is not possible to represent a set of products (with price and description) within the content of an individual page.
- Entering RDF triples is tedious since users need to know the concepts underlying the ontology. Furthermore, annotated markup is less readable than plain text.

The approach provides mechanisms to “associate arbitrary RDF with a wiki page”. This works for the definition of new RDF statements and additionally for imported ones. For instance, it is possible to import an existing ontology (e.g., RDFS) to a wiki page (i.e., the page's markup represents the ontology afterwards). This allows users to change an ontology according to the wiki way [LC01], quick and easy, by simply modifying the wiki text. However, the author also stated that it is quite difficult to directly change the RDFS without any tool support (e.g., by means of an ontology editor). In Hybrid Wikis the schema (types, attribute definitions, and constraints) is represented separately and changing it is supported by an UI.

Furthermore, in Kaukolu a wiki page can represent more than one subject. In particular, it is possible to mark individual sections within a page as subjects. By doing so, large sets of objects (e.g., a table of product descriptions) can be represented on a single page without the need to create an individual page for each object (e.g., one product per page). In Hybrid Wikis it is not possible to merge multiple individuals on one page²⁸. This is due to the fact that a page's URL serves as a conceptual identity in our model (cf., Section 3.1.2), similar to URIs in Semantic MediaWiki. Therefore, in Hybrid Wikis showing a set of products on a single page is only possible either by using an embedded custom (table) view (cf., Section 3.4.2) or by creating a semi-structured table within the built-in content. In the latter case, with little effort the table can be transformed into fully-fledged objects and an embedded custom table view showing them (cf., Section 3.3.2.5). Additionally, a custom table view allows to create a new object according to the view filter with one click and supports in-place editing for all shown instances. This makes it convenient for users to create and modify structured objects (e.g., products) without the need to leave the current context.

In order to make annotated content more readable in Kaukolu URI namespaces can be hidden by defining aliases for resources and predicates. Based on these aliases, autocompletion supports the user in annotating the wiki content (cf., Figure 5.13). Even if aliases improve the readability for authors, this “often leads to awkward sentences”. This is particularly true when subjects are declared within the content (cf., Figure 5.13). In Hybrid Wikis structured

²⁷<http://kaukoluwiki.opendfki.de>; visited on February 10th 2012.

²⁸Even if record values can be considered as anonymous objects, they can only be accessed via their owning content object.

and unstructured content are clearly separated, therefore content is always human-readable even in edit views.



Figure 5.13.: Kaukolu’s page editor showing suggestions based on imported ontologies according to [Ki06].

5.5.6. Artificial Memory

In [LOZ04, Lu05, Lu09] Artificial Memory²⁹ is introduced, a semantic wiki for personal and organizational knowledge management. The authors identified some drawbacks regarding annotations and ontologies. In particular, they stated that

- ontologies provided as annotated meta data lead to denormalized data since ontology pages are separated from instances pages,
- it is difficult to keep concepts and instances synchronized since authors have additional maintenance efforts,
- information is only partly annotated, and
- the connection between a document and the ontology cannot be made explicit.

The approach consciously tries to avoid the annotation of text strings due to potential risk of redundancies and denormalization of data. This is achieved by storing information as chunks, which are later serialized to hierarchically (tree-like) structured documents (cf., Figure 5.14). Therefore, Artificial Memory inverts the process of document creation “by first writing and semantically networking information chunks and only afterwards serializing those into documents”. By doing so, documents are always represented by meta data and the relationship between document and ontology can be maintained more easily and consequently stays consistent. Additionally, redundancies are avoided since information chunks (e.g., parts of the document’s hierarchy) or complete documents can be reused in other contexts by embedding them. Furthermore, a triple browser provides means for better navigation and facilitates the direct manipulation of structured information. When entering or modifying triples autocompletion is provided and consistency is checked in real-time.

Similar to Hybrid Wikis Artificial Memory

²⁹<http://www.artificialmemory.net>; visited on February 10th 2012.

- avoids the usage of annotations by providing a graphical user interface,
- facilitates the consistent use of terms and structures by means of autocompletion,
- reduces information redundancies by supporting embedded views, and
- allows the direct manipulation of structured instances within embedded views.

The main difference is that users in Artificial Memory are forced to structure information in order to create a document. This is due to the fact that a document is represented by a set of serialized information chunks only. In contrast, in Hybrid Wikis a content object can consist of both kinds of information, structured and unstructured. Therefore, wiki authors have the freedom to structure their objects.

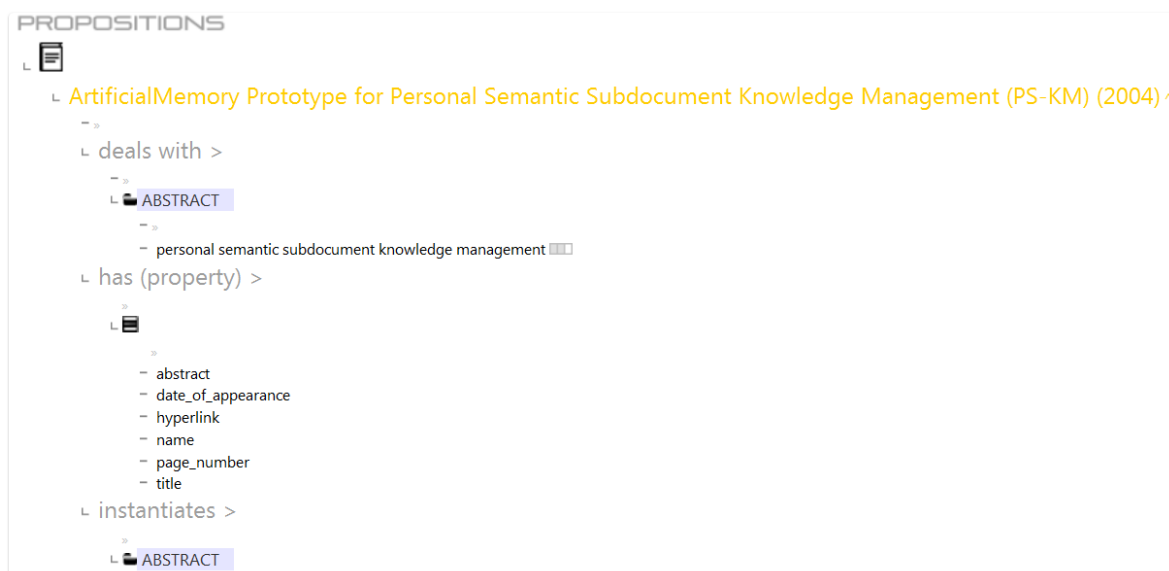


Figure 5.14.: Hierarchical document in Artificial Memory showing the triples of a paper entitled *ArtificialMemory Prototype for Personal Semantic Subdocument Knowledge Management*.

5.5.7. HYENA

In [RK06, Ra08, Ra10a, Ra10b] the HYENA platform is introduced as a mixture of RDF editor and wiki, that is, it combines structured and unstructured data. The approach follows the idea of incrementally structuring the wiki content (e.g., by tagging wiki pages), but the structure has a more complementary character and is considered to be optionally added afterwards. Even if the platform is based on the Semantic Web stack the focus is not to use RDF as knowledge representation but rather as standardized data structure.

HYENA can be used offline and online. Offline it is an Eclipse project (cf., Figure 5.15) which consists of directories containing two kinds of content items, files and RDF repositories. A repository can either be an RDF file or a reference to a central database. Wiki pages and their contents are also stored in these repositories. Therefore, a repository can contain both,

5. Related Work

structured and unstructured data. Additionally, for each data a description is stored. This description is used by so called inspectors in order to decide how data is presented in edit views.

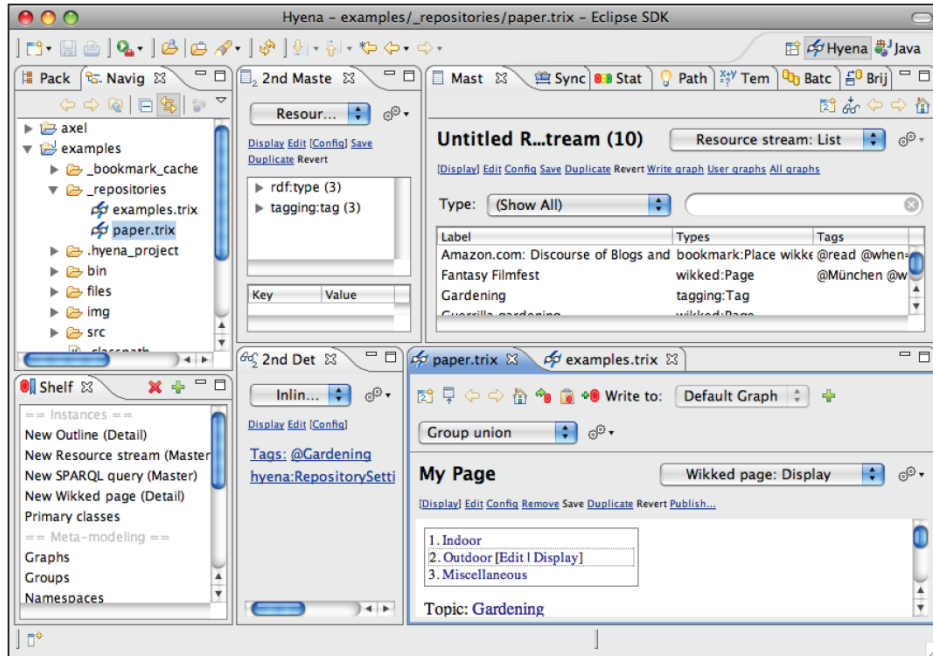


Figure 5.15.: HYENA's Eclipse-based offline RDF editor according to [Ra10b].

Repositories created offline (i.e., by means of the Eclipse desktop application) can be published to the web. Each repository is displayed as a web site (cf., Figure 5.15). If the repository represents a wiki page, it is displayed in wiki style accordingly. Resources created online or offline can be synchronized and merged. In order to structure wiki content (e.g., creating links to resources or tagging pages) a special syntax is used in HYENA, but the annotation of content with meta data is not possible.

Compared to other semantic wikis HYENA provides wiki capabilities based on an RDF editor and does not try to integrate RDF into a wiki. RDF in this approach merely serves as a standardized storage format (primarily not intended to be used for reasoning purposes). Therefore, in HYENA there is no need to additionally annotate wiki contents with RDF statements. This is similar to Hybrid Wikis since structured data is stored separately. However, in Hybrid Wikis an own representation of structure is used instead of RDF. Another difference is the possibility to maintain resources not only in the web interface but also through an editor running on the client's machine as a desktop application.

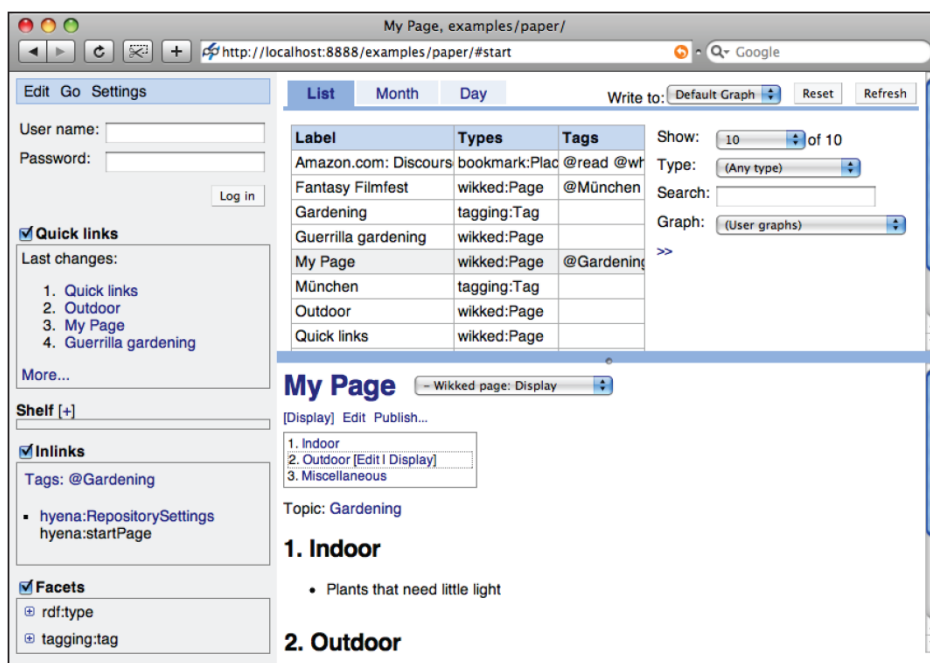


Figure 5.16.: HYENA’s web editor showing a wiki page selected in the list of available RDF resources according to [Ra10b].

5.5.8. IkeWiki

IkeWiki [Sc06, SGW05, SWG06]³⁰ and its derivation SWiM [La07, La10, La11, LK06]³¹ are semantic wikis supporting collaborative knowledge engineering. The main purpose of these approaches is to lower the technical barrier for users in creating meta data. This is mainly achieved by

- providing an interactive interface for annotations,
- supporting authors with (ontology-based) suggestions (e.g., link and page types),
- rewarding annotations with better presentation, navigation, and search capabilities,
- hiding advanced features from simple users, and
- supporting different stages of formalization (e.g., informal texts and formal ontologies).

While these goals are quite similar to the design principles underlying Hybrid Wikis, the main difference is that annotations in IkeWiki are associated with elements in the page’s content. Even if the wiki text and annotations are separated (i.e., meta data and wiki markup are not syntactically mixed) authors need to change to a special annotation mode (cf., Figure 5.17) in order to specify meta data. In Hybrid Wikis textual content and structure are clearly separated from each other but can be maintained together.

³⁰<http://sourceforge.net/projects/ikewiki/>; visited on February 17th 2012.

³¹<http://kwarc.info/projects/swim/>; visited on February 17th 2012.

5. Related Work

Furthermore, the approach aims at reusing existing knowledge sources, primarily articles from the Wikipedia encyclopedia. Therefore, it relies on the compatibility with the wiki markup underlying the Wikipedia platform in order to allow users to import articles from there.

Another difference is that pages in IkeWiki can only be associated with types (and properties) available in the system, that is, with types according to the ontologies currently loaded in the wiki. Even if new types and properties can be created by extending these ontologies users always need to leave the current edit context before they can associate individual pages with that extensions. In Hybrid Wikis any label (type tag) can be used to type a content object, independent of already existing type definitions. Additionally, in IkeWiki some types cannot be removed directly by the users since they are determined by reasoning mechanisms and automatically assigned to the page. “For example, if a link from ‘Mozart’ to ‘Die Zauberflöte’ is annotated by ‘composerOf’, the system will automatically associate the type ‘Composer’ with the page describing ‘Mozart’, and this type cannot be deleted directly by the user”. In Hybrid Wikis type suggestions are used instead (cf., Section 3.3.1.3). Therefore, authors are urged but never forced to use a specific type.

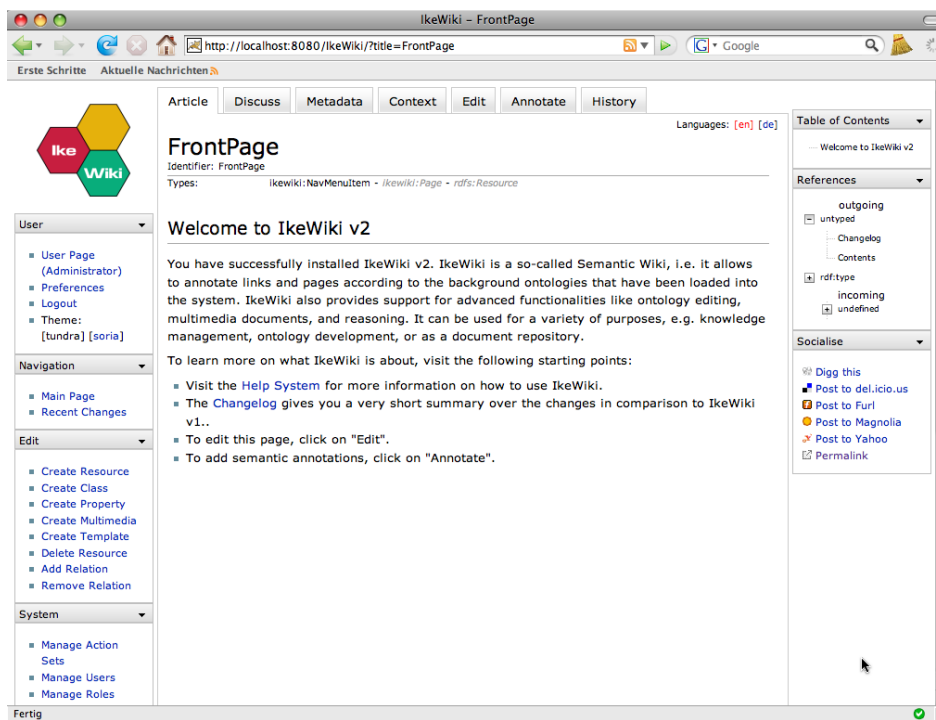


Figure 5.17.: The main window of IkeWiki showing navigation functionality on the left, wiki content in the middle, and meta data (e.g., incoming links) on the right. Source: http://www.wikimatrix.org/screenshots/screen_28_1.png; visited on February 19th 2012.

5.5.9. Makna

In [DST06, NS06] Makna is introduced as a wiki-based tool supporting knowledge engineering tasks. The approach focuses on facilitating the creation of semantic content for non-technical

users. This is mainly achieved by providing an interface that supports users to insert RDF statements in the wiki text. For instance, UIs are provided to insert single predicates or complete triples (even triples that do not appear in the pages). Additionally, these interfaces guide the users in entering data by supporting autocompletion based on the ontologies underlying the system. Consistency between the instances and the ontologies can always be guaranteed since users are rather enforced to enter valid statements only³².

In contrast to this, Hybrid Wikis consciously avoid enforcing validation rules, users are always free to add structures according to their business demands. Even strict constraints or rigid types can be softened by the wiki editors if they are disturbing. Furthermore, in Makna ontologies can only be modified by the wiki administrators, in Hybrid Wikis all wiki editors can create and change types, attribute definitions, and constraints.

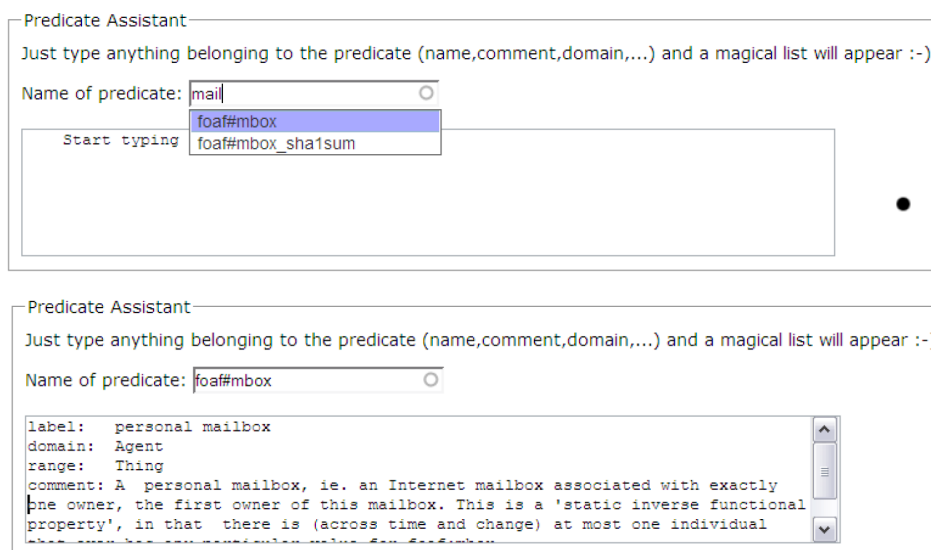


Figure 5.18.: Makna assistant to insert predicates in the wiki text according to [DST06].

5.5.10. KiWi

The EU-project KiWi³³ [Da08, Si09] is the successor of IkeWiki (cf., Section 5.5.8). In [Sc09] it is described as “a flexible and adaptable platform for building different kinds of Social Semantic Software, powered by Semantic Web technology”. In that article, the authors identified one main problem of existing Enterprise 2.0 platforms: even if they support different kinds of content objects, such as wiki articles and blog posts, changing an article to a post or creating new kinds of content types is not possible without changing the database. This is due to the fact that each content type is individually represented in the data model underlying the platform. That problem is addressed by so called “Content Versatility”. The main idea is to offer different views for the same content item without modifying the underlying data model. A content item is a piece of information (identified by URI) consisting of human-readable

³²The authors plan to support lightweight consistency in a future version.

³³<http://www.kiwi-project.eu>; visited on February 17th 2012.

5. Related Work

content and associated meta data. A view is an extension (called “KiWi Application”) to present content items to users in a specific way, for instance as a wiki page (cf., Figure 5.19). The complete core data model is represented by content items, tags, and triples.

User-defined tags can be related to content items again. “For example, the content item that describes ‘Mickey Mouse’ could be tagged with the label ‘Mouse’, thereby associating it with the content item describing ‘Mouse’ (the animal). The tagged content item would be ‘Mickey Mouse’, the tagging content item would be ‘Mouse’, and the tag label used for tagging would be ‘Mouse’, which is a tag label of the content item ‘Mouse’”.

meta data is represented in RDF triples in the KiWi system. In order to maintain additional information, such as versions or transactions, these triples are extended accordingly. Since it is not required to use a schema definition in RDF the data model can be changed at runtime.



Figure 5.19.: A semantic wiki page created based on the KiWi platform according to [Sc09].

KiWi is quite similar to Hybrid Wikis since both approaches provide a technological infrastructure for building social software applications supporting generic services in the core, such as versioning, tagging, and information structuring (based on a small set of structuring concepts). These services are orthogonally applicable to any kinds of content objects in the core or implemented within extension plugins, that is, KiWi Applications or Tricia plugins respectively. The difference is that content objects in Hybrid Wikis are individually represented in the data model, for example wiki page and blog post are both individual entities. This means a wiki page cannot be displayed as blog post as it would be possible in KiWi. Additionally, a new content type in Tricia can only be created by implementing a new entity type (e.g., tweet) which requires changing the database accordingly. In KiWi new content types can be introduced by programmatically extending the data model and implementing corresponding extension views, but without modifying the database.

Since tags in KiWi are treated as content items it is possible to tag tags. This enables meta modeling, which is consciously avoided in Hybrid Wikis since content objects, types, and tags are realized as different concepts. Another difference is that the approach tries to semi-automatically (i.e., in interaction with the user) extract meta data from the content.

In Hybrid Wikis information can only manually be structured. Furthermore, KiWi provides the possibility to create custom personalized views of content items based on meta data. In Hybrid Wikis meta data can be used to create custom views embedded in the content (cf., Section 3.4.2). Additionally, the presentation of individual content objects can be changed (e.g., hybrid view, structured view, cf., Section 3.4.1) but is the same for all users (i.e., cannot not be personalized individually).

5.5.11. Rhizome

In [So05, So06] Rhizome is introduced as a framework to built web application, such as semantic wikis, personal notes, and discussion forums.

The approach particularly addresses two problems:

- Create semantically enriched content maintainable even if it is copied or modified.
- Ease the use of Semantic Web technology.

The approach provides an own markup language (ZML) similar to wiki text but additionally allows to define semantics by using formatting conventions (cf., Figure 5.20). Semantic statements are translated to and stored as RDF triples but presented to the user in a simplified format (RxML).

Hybrid Wikis differ from this approach since structure is not embedded in the wiki page content. Furthermore, Rhizome simplifies the appearance of RDF elements by introducing own languages. However, users still need to learn these new languages, especially in a business context this is challenging.

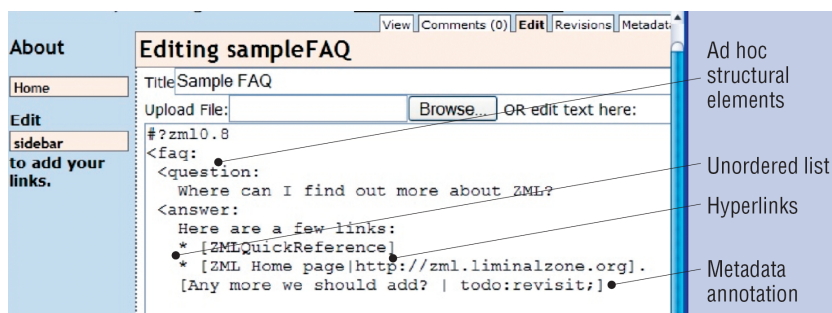


Figure 5.20.: Editor provided by Rhizome showing semantic annotations in its own markup language (ZML) according to [So05].

5.5.12. SHAWN

In [AA05, Au05c, Au05b, Au10] SHAWN (Structure Helps A Wiki Navigate)³⁴ is introduced as a semantic wiki prototype application facilitating information structuring for users. One objective of this approach is to keep the creation of structured content simple. This is addressed by

³⁴<http://sourceforge.net/projects/wiksar/>; visited on February 25th 2012.

introducing colon-separated key-value pairs which can be embedded in the page content (cf., Figure 5.21). A value is either a literal or a link. These key-values pairs are interpreted and stored by the system as RDF triples when the page is saved. Some keys have a specific meaning. For example, the key *InstanceOf* can be used to express that a page is an instance of the type given by the pair's value, that is, by using the *InstanceOf* key the page gets typed. In particular, these keys are used by the wiki engine in order to facilitate the navigation in the wiki (e.g., by providing breadcrumbs according to concept hierarchies), showing inverse (contextualized) links, and displaying sample values of shared properties from other pages (cf., Figure 5.21). This way, users instantly benefit from structuring contents.

In [Au10] the authors pointed out that “the SHAWN wiki is not meant to be a full fledged RDF/OWL editor”. This means, although an ontology emerges from the structured wiki pages it is primarily used to improve navigability in the wiki, additionally the structure can be exported to OWL. But it is not possible to apply that ontology to the data in this approach. For instance, it is not possible to consolidate data according the underlying schema or to check the data's consistency. Furthermore, data in SHAWN needs to be structured manually by inserting textual key-value-pairs in the page content. Additionally, there is no assistance for links creation since all words in the content (even the keys and values) are interpreted as links when the page is displayed.

HomePage | SiteMap | RecentChanges | RelShip | AllPages

HomerSimpson

- Something > LivingThing > SomeOne : HomerSimpson
- PartOf : TheSimpsons
- WorksAt : PowerPlant
- LivesIn : Springfield
- InterestsIn : DuffBeer

Homer Jay Simpson (voiced by Dan Castellaneta), is one of the main characters in the animated television series; TheSimpsons.

Homer works in the Springfield Nuclear PowerPlant, although “working” in this case refers to dozing and eating doughnuts. He spends a great deal of his time at Moe’s Tavern with his lifelong friends BarneyGumble, Carl, Lenny, and MoeSzyslak, the bartender.

```
InstanceOf: Someone
PartOf: TheSimpsons
WorksAt: PowerPlant
LivesIn: Springfield
InterestsIn: DuffBeer
```

Homer Jay Simpson (voiced by Dan Castellaneta), is one of the main characters in the animated television series; TheSimpsons.

HomerSimpson

SideBar

ForwardLinks

- BartSimpson (ChildOf)
- MargeSimpson (SpouseOf)

Also SomeOne as InstanceOf:

- BarneyGumble
- BartSimpson
- MargeSimpson
- MisterBurns
- MoeSzyslak
- WaylonSmithers

Also TheSimpsons as PartOf:

- BartSimpson
- MargeSimpson

Figure 5.21.: Editor provided by SHAWN using colon-separated key-value pairs for information structuring embedded in the page content according to [Au05b].

5.5.13. RISE

In [De05a, KHD05, KT04] the project RISE is introduced focusing on “reuse of software engineering knowledge supported by semantic wikis”. In particular, the authors show the

applicability of semantic wikis in the domain of requirements engineering. An ontology is used to represent the different types of documents (e.g., templates for user stories, actors, and use cases) and their connections. Wiki templates capture the structure and the relationship between pages and represent instances of these document types. Changing a template results in changing the underlying ontology. Additionally, semantic annotations can be used for checking the consistency.

Furthermore, in [De05a, De05b, KHD05] the authors introduce a paradigm called “Wikitology”. This paradigm means that it is possible to “semi-automatically derive an ontology” from the pages in the wiki (i.e., from the templates and the links between the pages). Since the ontology always reflects the knowledge in the wiki the ontology can be considered as self-adapting.

The idea of a self-adapting schema derived from wiki page instances in RISE is similar to the implicit schema used by Hybrid Wikis (cf., Section 3.2.2). However, the RISE approach relies on a set of naming conventions in using templates. These conventions are necessary to automatically determine the current ontology from the wiki pages. Additionally, the annotation of wiki content and the configuration of templates is not supported by a tool, that is, need to be performed manually.

5.5.14. SemperWiki

In [OBD06, Or06a, Or05] the authors characterize personal wikis as “simple and lightweight” by offering only a limited set of features. Furthermore, they support textual (linkable) notes, integrate desktop data, and provide full text search capabilities. But usually they are not intended for collaborative work. Semantic wikis are described as means for annotating wiki text to improve navigation and to offer (“simple or powerful”) querying capabilities. The introduced approach, namely SemperWiki³⁵, combines both personal information management and semantic wikis.

SemperWiki is an open source desktop application for Gnome³⁶. It supports pages using an own simple wiki markup language to express key-value pairs, as depicted in Figure 5.22 (e.g., *dc:topic "Explanation"*). Values can either be literals or links. These pairs are mapped to RDF statements using the page’s URL as the subject. Based on these statements, the system suggests related pages. Unlike Hybrid Wikis, SemperWiki is built for single user usage.

5.5.15. SweetWiki

In [BG06, Bu11a, Gh07] SweetWiki³⁷ is introduced as a wiki that combines semantic technologies with social tagging mechanisms [GH05]. The main idea underlying this approach is to derive an ontology (represented as RDFS) from the wiki pages being tagged. Users merely can add and remove tags to or from pages instead of using the ontology’s formal concepts directly. Each tag is mapped to a corresponding class in the ontology. New introduced tags (i.e., tags which are not yet represented as concepts in the ontology) can be integrated in the

³⁵<http://www.eyaloren.org/semperwiki.html>; visited on May 8th 2012.

³⁶<http://www.gnome.org>; visited on February 25th 2012.

³⁷<http://www-sop.inria.fr/teams/edelweiss/wiki/wakka.php?wiki=SweetWiki>; visited on February 25th 2012.



Figure 5.22.: SemperWiki editor showing keys (e.g., *dc:author*) and values (e.g., "Explanation") in the wiki page content according to [Or05].

ontology by using a specific editor. This editor is accessible to all users and additionally provides means to create relationships between tags (e.g., hierarchies) and to merge tags (e.g., in case of synonymous). Based on these relationships, tags are suggested (e.g., compatible tags according to the specified hierarchy) when the user enters keywords (cf., Figure 5.23).

In SweetWiki a tag not necessarily needs to be assigned to an ontology class. This is similar to Hybrid Wikis since type assignments (type tags) and types can be specified independently. However, the main difference of both approaches is that in Hybrid Wikis a tag can explicitly be used as a type, that is, a user can explicitly make a statement about a page's type by using type tags. This way, the user is aware of typing a page.

Furthermore, in contrast to Hybrid Wikis SweetWiki is using a different way to implement content objects, called "The Wiki Object Model". This means, the concepts underlying the system, such as wiki pages, documents, and files, are represented in a separate ontology. This ontology is maintained by the wiki developers and corresponding meta data is integrated in the pages' contents. By doing so, it is for example possible to specify queries accessing that declarative content model.

5.5.16. SOBOLEO

In [BSZ07, ZB07] SOBOLEO (Social Bookmarking and Lightweight Engineering of Ontologies)³⁸ is introduced as a tool that combines social bookmarking with collaborative ontology engineering. Users can annotate web resources by means of tags. These tags can be organized in an ontology, for instance it is possible to create tag hierarchies. Additionally, tags can pro-

³⁸<http://www.soboleo.com>; visited on February 25th 2012.

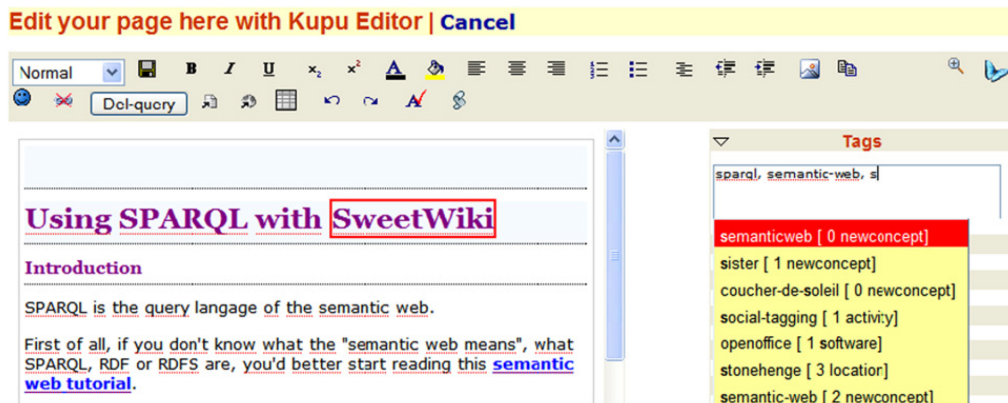


Figure 5.23.: Editor provided by SweetWiki showing tag suggestions as the user enters keywords according to [Bu11a].

vide a textual description. Tag-based ontologies improve information retrieval in SOBOLEO and enable editing of a resource's tags supported by autocompletion (cf., Figure 5.24).

SOBOLEO is quite similar to SweetWiki (cf., Section 5.5.15). The difference between both approaches is that SOBOLEO is rather intended to organize existing web resources. However, the main difference compared to Hybrid Wikis is that a resource cannot explicitly be typed by using a tag.

Webdokument Taggen

URL:
Titel:
Themen:
Aktuelle Themen: Wikis

- social software
- hybrid wiki
- semantic wiki
- enterprise 2.0
- tricia

Figure 5.24.: Editor provided by SOBOLEO showing tag suggestions as the user enters keywords.

5.5.17. KnowWE

In [BP08, BRP07a, BRP07b, BRP11, RBP08, Re10] KnowWE (Knowledge Wiki Environment)³⁹ is introduced as a semantic wiki focusing on the development and the consumption of knowledge in problem-solving contexts. The approach allows to represent, maintain, and use an ontology (with classes and user-defined properties) by means of annotations in the page content. Additionally, it is possible to embed problem-solving descriptions (e.g., derivation rules or decision trees necessary to calculate solutions) in the content represented in a specific syntax (cf., Figure 5.25 (1)).

In order to find a solution users are asked questions in an interview session. A question is related to a concept of the ontology and can be defined by annotating a page's content. Therefore, answering a question can be considered as temporarily (i.e., during the interview session) instantiating an ontology class (cf., Figure 5.25 (2)). Based on the user's answers (i.e., the user's findings) and the embedded problem-solving descriptions, possible solutions are derived and suggested to the user (cf., Figure 5.25 (3)).

In contrast to Hybrid Wikis, KnowWE is designed for problem-solving purposes and represents a tool for building decision-support systems.

5.6. Corporate Semantic Web

In [Pa09] the working plan for a project named Corporate Semantic Web is introduced. The authors propose a lightweight approach to ontology engineering, semantic collaboration, and semantic search. In particular, the approach aims at

- facilitating the ontology creation process by modularizing and versioning ontologies,
- enabling knowledge workers to change ontologies without the need to modify the underlying formalisms, and
- combining searches for semantic data and none semantic data using tagging, text mining, and personalization techniques.

The Corporate Semantic Web approach facilitates the management of structured data. However, this is planned to be achieved by using a lightweight ontology editor, which is different compared to Hybrid Wikis.

5.7. Freebase

Freebase⁴⁰ is an open graph database to collaboratively create “structured general human knowledge” for public use [Bo07, Bo08, BCT07]. The ideas underlying this approach are quite similar to those of Hybrid Wikis, that is, it aims at

- collaboratively and incrementally structure information,

³⁹<http://sourceforge.net/apps/mediawiki/knowwe/index.php>; visited on February 17th 2012.

⁴⁰http://wiki.freebase.com/wiki/Main_Page; visited on February 2nd 2012.



Figure 5.25.: Editor provided by KnowWE showing knowledge used for problem-solving purposes and recommended solutions according to [BRP11].

- facilitating information structuring by providing an easy-to-use web UI, and
- supporting structured queries efficiently using suitable storage and index structures.

Basically, the type system underlying Freebase consists of domains, topics, data types (e.g., string or number), properties, and types (cf., Figure 5.26). Domains correspond to spaces, topics to content objects, and properties to attributes in Hybrid Wikis. The other concepts (i.e., data types, and types) correspond to each other. Even if both approaches follow the same principles and provide almost the same set of concepts the main difference is that data and schema in Freebase are coupled more closely. That is, in order to add new attributes to a topic it is required to extend the schema before. Furthermore, it is not possible to relate attributes to topics without having a type assigned. This means, attributes can only be related to exactly one type. Therefore, a topic's author inevitably needs to act as a tailor (cf., Section 3.2.1) in order to structure information which is consciously avoided in Hybrid Wikis. Additionally, in Freebase the schema can only be adapted by its creator. This is to prevent other users from breaking the tailors' intention of the schema, but obviously reduces the degree of flexibility in collaborative information structuring. Moreover, a user not being the owner (i.e., creator) can only improve the schema by duplicating it completely.

Finally, attributes are only suggested if a type is assigned. For these reasons, we consider Freebase as a more heavyweight approach in structuring information than Hybrid Wikis.

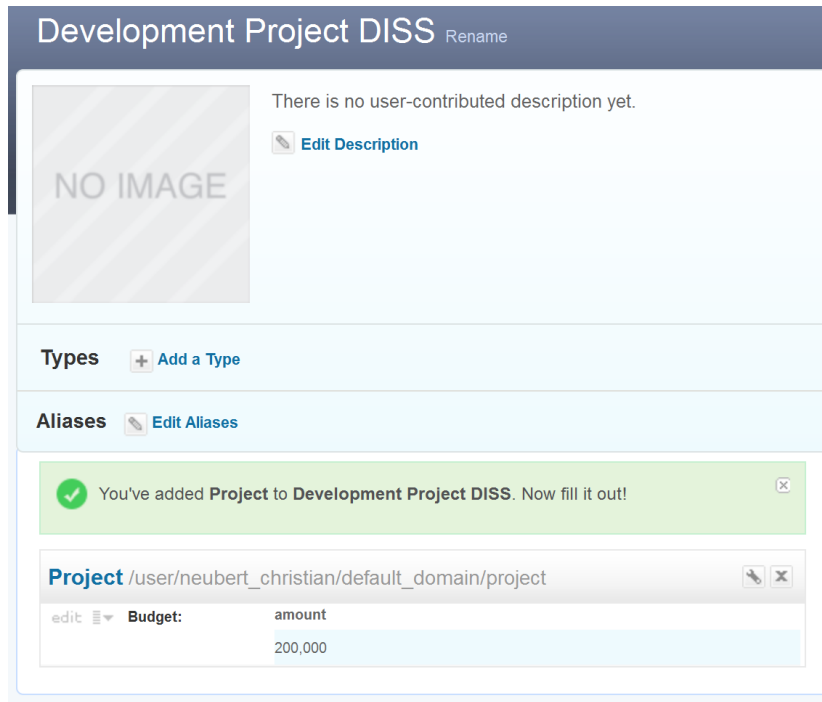


Figure 5.26.: A Freebase topic *Development Project DISS* with attribute *Budget* typed as *Project*.

5.8. MoKi

In [Gh09, Ca11, Ro09, Ro08] MoKi (Modelling wiKi) is introduced as “a wiki-based tool for enterprise modeling” built on the basis of Semantic MediaWiki (cf., Section 5.5.1). The approach’s primary purpose is to collaboratively model the constituents of an enterprise, in particular domain objects, business processes, and competencies of employed people. Business users and domain experts collaboratively create such a model. In order to not require these actors to learn wiki syntax structured data can be entered by using wiki templates and forms. Depending on the page’s type (i.e., domain object type) predefined templates offer for example properties intended to be filled out when editing the typed page (cf., Figure 5.27). Additionally, wiki templates are utilized to transform unstructured content in a formal model.

With Hybrid Wikis we focus on facilitating collaborative data and information management within enterprises. Our approach is not specifically optimized for modeling information of a specific domain, such as the constituents of an enterprise⁴¹. Models in Hybrid Wikis guide and support business users in their day-to-day information management activities, emerge bottom-up as a by-product, and continuously evolve over time.

⁴¹In Hybrid Wikis models emerging through data management can also be used to represent enterprise constituents (cf., [Bu09], Section 6).

Modify concept: Conference

Annotations

Description:

Synonyms:

Hierarchical Structure

Is a:

Is part of:

Properties

Property:

Property target:

Figure 5.27.: MoKi editor showing a wiki page with textual description, hierarchical structure, and properties according to [Gh09].

5.9. SnoopyDB

In [GZS11, Gal0b, ZGS10, ZG10] the research prototype SnoopyDB⁴² is introduced. The approach addresses problems in information structuring identified in literature, such as proliferation and divergence of schemas, “by facilitating structure and developing a common schema by providing recommendations”. The approach particularly focuses on recommending attributes and values to users while entering data. Recommendations guide users in order to align entered information to existing structures. They are dynamically calculated based on the current editing context (i.e., depending on the attributes and values assigned to the resource currently edited) and the structures (i.e., key-value pairs) already available in the system. Therefore, existing structures can be considered as an implicit schema. This schema is also referred to as “self-adapting”. That means the schema is continuously evolving since all changes to attributes and values impact the algorithm calculating the recommendations and thereby users (probably) apply (recommended) structures commonly used.

Recommendation for attribute keys in SnoopyDB are similar to attribute suggestions in Hybrid Wikis. That is, they are suggested to be used in addition to the attributes already assigned. Furthermore, SnoopyDB supports autocompletion for attribute keys and values. Literals as well as links are suggested while a user enters a value. Similar to Hybrid Wikis recommendations and autocomplete results are calculated based on the current context of the edited resource and the (implicit) schema. SnoopyDB also provides suggestions regarding data types. That is, when a user enters a literal that can be interpreted as a number, the data type number is suggested to be applied. Furthermore, a data type is suggested for a value if the majority of other attributes (having the same key) use a different one.

Even if both approaches follow quite similar ideas (e.g., suggestions or implicit, adaptive schema) the main difference of SnoopyDB compared to Hybrid Wikis is that types cannot

⁴²<http://dbis-informatik.uibk.ac.at/168-0-SnoopyDB.html>; visited on February 22nd 2012.

be specified at all. This makes it difficult to generate tabular overviews of resources covering certain kinds of things. Furthermore, in that approach it is not possible to explicitly define a schema, that is, attributes cannot be bound to a type or integrity constraints be specified.

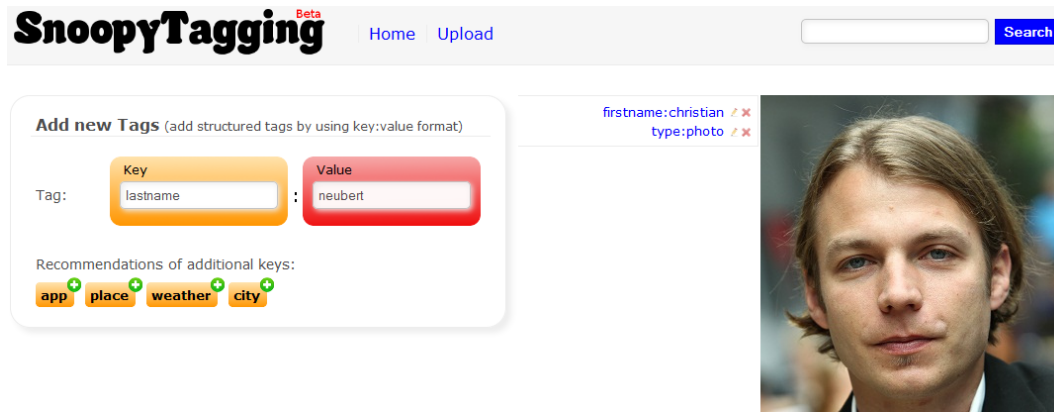


Figure 5.28.: Editor provided by SnoopyDB supporting structured tags (in key:value format) applied to a Flickr photo (<http://www.flickr.com>; visited on February 22nd 2012).

5.10. Social Infobox

In [HGT11] the prototype system Social Infobox⁴³ is introduced. The system supports “social property tagging”, a method to freely add attribute-value pairs to wiki-like resources without using an explicitly defined schema. In that article, a user study showed that property names were shared among different resources which the authors interpret as the emergence of implicit types. Furthermore, the system provides property suggestions based on an analysis of attribute cooccurrences (cf., Figure 5.29). Social Infobox and Hybrid Wikis follow similar ideas, such as bottom-up schema evolution and attribute suggestions, but there are two important differences:

- By using type tags Hybrid Wikis enable users to explicitly make a statement about the type. Since attributes can be used independently, this does not limit the flexibility.
- Besides the analysis of attribute cooccurrences, attribute suggestions in Hybrid Wikis are based on the combinations of type tags and attributes used in the system. This means that not only the frequency of occurrence is considered, but additionally, attributes are preferred when they occur together with many or all of the type tags.

⁴³<http://hamlab0.hpcc.jp/sdow>; visited on February 22nd 2012.

Subject: [Development Project DISS](#)

As a: [\[subject\]](#) [\[property\]](#) [\[object\]](#) See also: [\[Wikipedia\]](#) [\[Google\]](#) [Delete this resource](#)

Development Project DISS

[Cancel](#) | [View history](#) | [Show examples](#)

no data

Literal Resource(s) [Update](#)

Budget 200.000

New property:

Literal Resource(s) [Add](#)

[Add property to Development Project DISS](#) [Add](#) [Compare resource with Development Project DISS](#) [Compare](#)

[made](#) [coauthor](#) [presentationIn](#) [title](#) [author](#) [webpage](#) [seeAlso](#) [proposer](#) [start](#) [topics](#) [of](#) [interest](#) [issue](#) [history](#) [Location](#) [affiliation](#) [scopeOfApplication](#) [purpose](#) [presentations](#) [background](#) [currentProject](#) [approach](#) [motivation](#) [member](#) [Primary](#) [objects](#) [electoricTown](#) [analogy](#) [Designed](#) [for](#) [birth](#) [date](#) [triple](#) [store](#) [extension](#) [authorize](#) [workPlace](#) [relatedPerson](#) [pronounce](#) [faceImage](#) [principles](#) [Affiliation](#) [chair](#) [favoriteAlgorithm](#) [typesOfMachineLearning](#) [is-a](#) [algorithm](#) [laboratory](#) [Links](#) [between](#) [related](#) [project](#) [interlinked](#) [dataset](#) [general](#) [requirement](#) [component](#) [originality](#) [capital](#) [datasets](#) [URI](#) [pattern](#) [presentationType](#) [presentationTime](#)

[Takayuki](#) [Goto](#) [birth](#) [date](#) [seeAlso](#) [Yutaka](#) [Matsuo](#) [Location](#) [Chris](#) [Bizer](#) [abstract](#) [SVM](#) [Linked](#) [Data](#) [LDA](#) [SIOC](#) [SVD](#) [pLSA](#) [Extracting](#) [Semantic](#) [Rela...](#) [End-User](#) [Programming](#) [and...](#) [Enabling](#) [Ontology-based](#) [...](#) [Mapping](#) [Tweets](#) [to](#) [Confer...](#) [Computing](#) [FOAF](#) [Co-refere...](#) [What](#) [does](#) [It](#) [Look](#) [Like...](#) [Masahiro](#) [Hamasaki](#) [SDoW2009](#) [DBpedia](#) [Semantic](#) [History:](#) [Toward...](#) [Multiple](#) [Personalities](#) [o...](#) [Reactivity](#) [and](#) [Social](#)

Figure 5.29.: Editor of Social Infobox showing a resource with properties and suggestions.

5.11. TWiki

In [Am10, Ch08b, LFL05, JH11, RG02, RS04] the open source project TWiki⁴⁴ and its fork Foswiki⁴⁵ are described as structured wikis combining the advantages of wikis and database systems. The approach allows users to attach data records to wiki pages. These records can be defined as a template and embedded (by reference and not copied) in the page's content. An embedded template is normally shown as a web form (i.e., as a set of keys and input fields) when the page is displayed. Templates can be defined within the content of a wiki page using a mixture of HTML, Javascript, and a special syntax (cf., Figure 5.30). By means of embedded searches tabular views can be produced showing objects for example using the same type of template.

Although the schema in TWiki can be changed at runtime the effort is considerably higher than for Hybrid Wikis since the definition of a template needs to be configured manually. Additionally, the usage of a special wiki syntax and web development elements is required, which is comparable to programming the schema.

⁴⁴<http://twiki.org>; visited on February 22nd 2011.

⁴⁵<http://foswiki.org>; visited on April 27th 2011.

5. Related Work

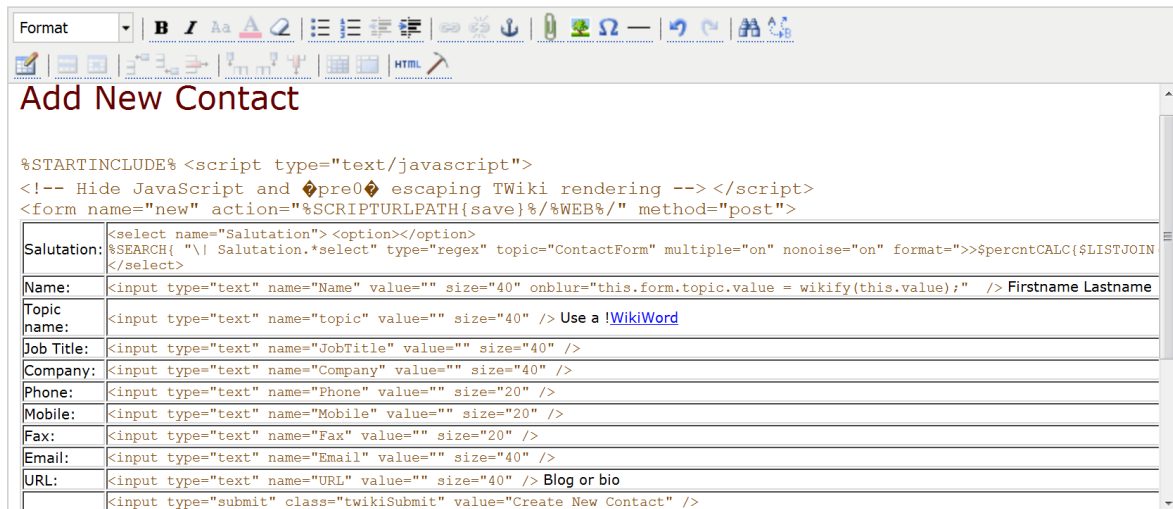


Figure 5.30.: Editing a template for contacts in TWiki.

5.12. DBWiki

In [Bu11c, Bu11b, Pe11] DBWiki⁴⁶ is introduced as a structured wiki that combines the advantages of databases (e.g., scalability) and wikis (e.g., convenience). The data model underlying the approach is represented by a hierarchical *Extensible Markup Language* (XML)-like tree. This tree contains both, information about the schema and the data. The schema describes possible paths in that tree. The tree is stored by mapping its elements (i.e., data and schema) to concepts of a relational database. The data model (i.e., schema and data) can be changed by adding or removing subtrees.

Additionally, the approach allows users to create wiki pages containing unstructured content. By using an extended markup language users can interact with that tree-based data model by embedding queries in the wiki page content. By doing so, it is possible for example to create editable views (e.g., editable tables) based on XPath⁴⁷-like queries (cf., Figure 5.31). Changes in a view are propagated to the underlying data tree and finally are mapped to corresponding operations in the relational database (i.e., insert and update statements).

Even if data can be modified by means of editable views embedded in the page content the query underlying this view needs to refer to existing paths in the data tree. In consequence, this means that only nodes resulting from a query can directly be modified or deleted in the wiki pages. Changing the nodes' hierarchy (e.g., in order to create a new property) is only possible by using a specific tree browser. In Hybrid Wikis new properties and concepts can be introduced directly in the wiki pages. Furthermore, Hybrid Wikis provide means to build custom embedded (table) views referring to none existing types and properties (cf., Section 3.4.2). In DBWiki changes in the data model (i.e., in the data tree) lead to updates in the underlying relational database. This means database policies need to be considered which potentially makes the data migration process more complicate. In Hybrid Wikis no modifica-

⁴⁶<http://forum.idea.ed.ac.uk/idea/database-wiki>; visited on February 25th 2012.

⁴⁷<http://www.w3.org/TR/xpath/>; visited on May 9th 2012.

tion of the database structure (e.g., changing the structure of a database table) is needed in case of changing information structures.

To the best of the authors knowledge, DBWiki only allows to specify queries for structures in the data tree, querying for unstructured information (i.e., wiki page content) is not supported. But the way the data is stored in DBWiki allows to define queries over the data's history, which is different to Hybrid Wikis. However, it would be interesting to investigate how to support temporal modeling and temporal queries in future research. Especially in the context of enterprise architecture management (cf., Section 6.1.1) it is often required to define queries over the history of data (e.g., in case of modeling the current and the future state of an application landscape within the same wiki).

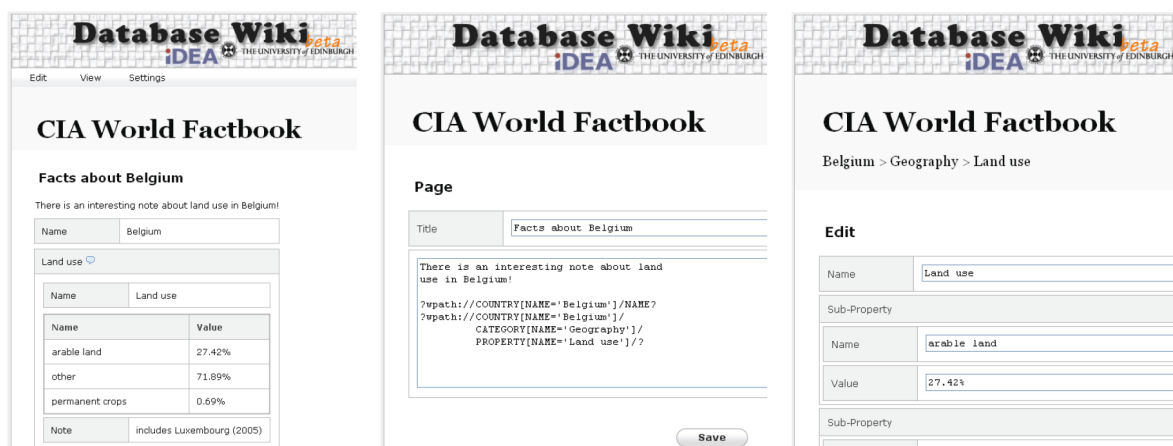


Figure 5.31.: Embedded queries and update forms in DBWiki's pages according to [Bu11b].

5.13. Summary and comparison with Hybrid Wikis

In the following, we briefly summarize the key findings of our literature analysis in order to highlight the distinguishing, innovative aspects of our approach and some commonalities. We roughly arranged the identified approaches in five categories regarding their key characteristics.

5.13.1. Semantic annotations

The first implementations of semantic wikis, that is, the combination of Semantic Web technologies [BL98] and wikis [LC01], emerged around 2004. Our literature analysis shows that this combination is frequently used in order to structure information in wiki systems, 17 of 28 of the identified approaches are semantic wikis. Early semantic wikis, such as the Platypus wiki [CCT04, TCC04], are more focused on the question how to bring both worlds (i.e., semantics and wikis) together in terms of concepts and technology. Therefore, many of them introduce semantic annotations as part of the wiki text (e.g., by using a special wiki syntax). This way, users are enabled to integrate and to use the full expressive power of the Semantic Web (e.g., in terms of modeling and querying capabilities) within a wiki. Today's most

prominent semantic wiki that supports structured data by using annotated wiki markup is Semantic MediaWiki [KVV06]. However, as described in this chapter researches identified some drawbacks (in particular for wiki authors) regarding the use of semantic annotations, such as additional maintenance efforts without having apparent benefits or worsened readability of the contents. Therefore, we consciously decided not to make use of semantic annotations in Hybrid Wikis, content and meta data are clearly separated.

5.13.2. Towards simplicity

Around 2006, these previously mentioned drawbacks were addressed by the ‘second generation’ of semantic wikis that try to find ways to

- lower the technical barrier for users in creating meta data (cf., Section 5.5.8),
- examine how users unfamiliar with semantic technologies can be helped and motivated to specify the many necessary typed links in a way that results in a useful web of annotations (cf., Section 5.5.2),
- “rapidly simplify the presentation and acquisition of instance data from and for end users” [ADR06] (cf., Section 5.5.4),
- facilitating the creation of semantic content for non-technical users (cf., Section 5.5.9), and
- keep the creation of structured content simple (cf., Section 5.5.12).

For example, this is achieved by hiding the conceptual expressivity of the underlying structuring language (e.g., RDF) from the users, introducing graphical user interfaces to manage annotations of content (e.g., by means of a graphical annotation toolbar), and rewarding users for providing structures (e.g., by offering improved navigation capabilities). Although these means mitigate the drawbacks of annotations all approaches differ from Hybrid Wikis in some major aspects, for instance:

- Some approaches enforce consistency (cf., Section 5.5.9) while others do not provide any mechanisms to validate structured data (cf., Section 5.5.12).
- When providing graphical UIs users are forced to enter data in a structured way (cf., Section 5.5.4) or they have to change to a special mode in order to provide structured content (cf., Section 5.5.2) or to modify the schema (cf., Section 5.5.8).
- Only certain persons (e.g., wiki administrators) are allowed to change the schema (cf., Section 5.5.9).

From our point of view, these restrictions result from the potential risk that schema and data diverge in case of user-initiated changes.

Besides semantic wikis, other approaches enable users to capture information in a structured way (e.g., in order to avoid information redundancies), such as wiki templates (cf., Section 5.2) or table extensions (cf., Section 5.3). There are two major drawbacks of these solutions:

- When copying templates (or tables) there is a potential risk that schema and data diverge over time.

- When using templates (or tables) by reference wiki page authors have to leave the current edit context in order to change to schema (i.e., the referenced template).

In Hybrid Wikis authors can implicitly change the schema by introducing new attributes on the instance level without leaving the current edit context. Additionally, Hybrid Wikis counteract structural divergence by guiding users towards a consistent usage of type, attributes, and constraints. Hybrid Wikis support context-dependent autocompletion for terms (e.g., attributes keys depending on the already assigned attributes) and provide suggestions to use schema elements (e.g., defined attribute keys) on the data level or to transfer frequently used data elements (e.g., data types) to the schema. Furthermore, Hybrid Wikis warn users if they violate a (explicit or implicit) convention and support them in finding inconsistencies. Inconsistencies can be harmonized by using consolidation techniques. By means of these mechanisms it is not necessary to force users to follow a prescribed schema or to prevent user-initiated schema changes.

Another observation is that tools and prototypes of the ‘second generation’ tend to use structures in two different ways. Either they use structured wiki pages to collaboratively create or (semi-) automatically derive a schema (e.g., an ontology) for different purposes (cf., Section 5.5.13) or they integrate a (pre-configured) schema in the wiki in order to improve specific system functions, such as navigation, authoring, or search capabilities (cf., Section 5.5.9). However, most⁴⁸ of these approaches unnecessarily miss the potentials of a smooth, mutually complementary integration of schema and data. Hybrid Wikis support both ways, that is, the derived schema helps to improve systems functions.

5.13.3. Structures for public use

Around 2008, DBpedia (cf., Section 5.4) and Freebase (cf., Section 5.7) are introduced. Both approaches focus on sharing and exchanging structures in the web. Due to the public use of structures, in Freebase changes to the schema can only be performed by the creators. For instance, only the creator of a type is allowed to add new attributes to it. Hybrid Wikis are built for the application in enterprise contexts in which the degree of trust between users is higher than in the public web. Therefore, in Hybrid Wikis it is not required to restrict changes to the schema also because users can register to get notified in case of modified structures (e.g., modifications to types) and or they can keep track of changes by means of the version history.

5.13.4. Social tagging

The ‘third generation’ of identified approaches emerged in the period from 2007 to 2011. These tools and prototypes are characterized by offering users rather a reduced set of lightweight structuring concepts than a formal, standardized description language. For example, Sweet-Wiki (cf., Section 5.5.15) and SOBOLEO (cf., Section 5.5.16) use resources (e.g., wiki pages) being tagged to derive an ontology, SnoopyDB (cf., Section 5.9) and Social Infobox (cf., Section 5.10) allow to freely assign simple key-value pairs to resources without using an explicitly defined schema. The latter tools guide users towards a consistent vocabulary by providing

⁴⁸Some approaches consciously decided to pursue only one direction.

key and value suggestions. Suggestions are calculated based on the key-value pairs assigned to the resources. However, since no types are provided users cannot define consistency rules. In the former approaches it is not possible to explicitly make a statement about a resource's type since tags are always interpreted as types. In Hybrid Wikis type tags are lightweight means to define an object's type and types can be used to specify integrity constraints.

5.13.5. Applicability in enterprises

Finally, a remarkable fact is that only two approaches are specifically developed for the application in enterprise environments, namely MoKi (cf., Section 5.8) and Semantic Enterprise Wiki (SWM+) (cf., Section 5.5.2). MoKi's allows to collaboratively model the constituents of an enterprise, but is developed for this purpose only. SWM+, as previously described, mitigates some drawbacks of Semantic MediaWiki by providing for example a graphical UI to enter semantic annotations for contents, but leaves some open (usability) issues (e.g., the synchronization of content and annotations).

It is not clear if the other approaches provide the functional capabilities essential for the application in an enterprise context, such as groups, access control, versioning, awareness, or if they are prototypes never intended to be applied in an enterprise environment but rather developed to answer specific research questions.

Our literature analysis shows the novelty of Hybrid Wikis. Although even other approaches pursue similar objectives, none of them supports user-enabled data and schema alignment as provided by Hybrid Wikis with for example different stages of rigidity, consolidation techniques, and suggestions.

In this chapter, we present the validation of Hybrid Wikis. In Section 6.1, we introduce selected case studies in six organization from different application domains. In particular, we sketch the business users' perceived benefits and highlight the information structures that emerged by using model visualizations inspired by UML class diagrams. To evaluate the quality of structures created with Hybrid Wikis, in Section 6.2 we present the results of a controlled experiment conducted with students at Technische Universität München. In Section 6.3, we introduce a generic tool that allows to quantitatively analyze structures built with Hybrid Wikis.

6.1. Applying Hybrid Wikis

In the following, we present the experiences gained with Hybrid Wikis in practice. We examine usage scenarios of Hybrid Wikis by presenting selected case studies from industry. In Section 6.1.1, we show how Hybrid Wikis are applied in the context of enterprise architecture management, in Section 6.1.2 Hybrid Wikis are used as an issue tracker, in Section 6.1.3 Hybrid Wikis support the management of course exercises, and in Section 6.1.4 it is analyzed how business processes of a medium-size enterprise can be documented and carried out with Hybrid Wikis. To illustrate the structures that emerged in these cases, we use UML-like class diagrams (cf., Section 3.4.1.8).

6.1.1. Wiki4EAM Community

Enterprise Architecture Management (EAM) is a challenging task modern enterprises have to face [LKL10]. This task is often addressed via heavy-weight and expensive EAM tools to collect, structure, visualize, and analyze architectural information, such as business processes,

applications, organizational units. A major problem in EAM is the mismatch between the existing unstructured information sources and the rigid information structures and collaboration mechanisms provided by today's EAM tools [Ma08]. Furthermore, the architectural knowledge is spread over all the different stakeholders in the company. Therefore, even the documentation of the current state of the *Enterprise Architecture* (EA) is difficult.

To mitigate these challenges, in December 2010 at Technische Universität München we established a community (Wiki4EAM¹ [MN11b]) of experienced enterprise architects from 25 large German enterprises (industry and the public sector) to pursue a lightweight, wiki-based approach to EAM [Bu09, Bu10, MN12]. The idea is to start with existing unstructured information sources (e.g., derived from spreadsheets and Office documents) captured as wiki pages and then to incrementally and collaboratively enrich these pages with concepts provided by Hybrid Wikis (i.e., attributes, types, and constraints) as needed for architecture modeling, visualization, and analysis.

Customizable in-browser visualizations are provided by the open source *System Cartography* (SyCa)-Tool² developed at Technische Universität München [BMS10]. The SyCa-Tool provides means to create maps visualizing enterprise landscapes based on arbitrary object-oriented information models. Both tools interact via REST-ful web services. From a user's perspective visualizations can directly be maintained within Hybrid Wikis. The general process of creating visualizations based on structured wiki pages is:

By means of the faceted search (cf., Section 3.4.3) users can drill down the search results to shape the set of structured objects underlying the visualization. This search (i.e., the query string) can be embedded in the content of wiki pages (cf., Section 3.4.2), additionally specifying some visualization-specific configuration parameters, such as the map type and the elements' colors. For instance, it can be configured to render a cluster map (cf., Figure 6.1) with outer elements (blue) represented by objects of type *organizational unit* and inner object (red) typed with *business application*. Furthermore, it can be specified which attribute key (e.g., *responsible for*) connects inner and outer objects in order to present objects of type *business application* nested in *organizational unit*. When displaying this page the embedded query is executed and from the structured objects of the result set a model (represented in Ecore format) is derived. This model, the instance data (i.e., structured wiki pages transformed to XMI), and the visualization-specific configuration are passed to the SyCa-Tool via a web service request. The SyCa-Tool answers this request by sending a generated map (in *Scalable Vector Graphics* (SVG) format) back to Hybrid Wikis.

Between December 2010 and March 2012 we conducted seven workshops together with the Wiki4EAM community members. In the initial workshop, the participants were introduced in the general idea and the core concepts underlying Hybrid Wikis by presenting some slides. Furthermore, we used a projector to demonstrate the handling of the software by using a small sample scenario from the EAM domain. For instance, we showed how to structure pages, create constraints, and embed visualizations. The projector-based introduction to Hybrid Wikis took about two hours in total (including questions from the audience).

The participants of the initial workshop saw the following potentials in applying Hybrid Wikis in their enterprises (extract from an informal discussion during the workshop):

¹<http://www.matthes.in.tum.de/wikis/sebis/wiki4eam>; visited on March 10th 2012.

²<http://www.matthes.in.tum.de/wikis/sebis/sycatool>; visited on March 10th 2012.

Wiki4EAM AG

What are you looking for?

» Wikis » IT-Landscape » Clustermap

View Details Versions Edit

Clustermap

Tags: [visualization](#) [eam](#) [map](#) [clustermap](#) [wiki](#)

Which organizational unit is responsible for which business application?

Headquarter			Subsidiary Hamburg	
Customer Satisfaction ...	Human Resources Sy...	A Document Management...	Worktime Management ...	POS System (Germany/Ha...
Data Warehouse	Financial Planning Sys...	Business Traveling Sy...	Monetary Transactions ...	Monetary Transactions ...
MIS (Management ...)	Accounting System	Online Shop	Price Tag Printing Sys...	
Customer Complaint Sy...	Costing System			

Figure 6.1.: Cluster map embedded in a wiki page provided by the SyCa-Tool based on models derived from structured pages in Hybrid Wikis

- Hybrid Wikis are well suited for the initial, collaborative documentation of the EA's current state since information can quickly and easily be structured and structures flexibly be adapted.
- Hybrid Wikis are a useful supplement to existing EAM tools. Models emerge bottom-up by focusing on contents instead of 'inventing' models and filling them with data. These emergent models can be used as an input for a conventional EAM tool in order to perform for example advanced planning tasks.

The participants saw the following challenges (extract from an informal discussion during the workshop):

- There is a potential risk of data chaos in scenarios with a large number of stakeholders, especially when using soft constraints only.
- It is difficult to integrate 'another' tool in the existing IT landscape, for example due to reasons of data migration efforts or costs in building interfaces to external data sources.

After the initial workshop the software (including some EAM demo data) was made available to the members. They used it either online hosted at Technische Universität München or

downloaded³ it for purpose of local installation. In the subsequent workshops, members of the community presented their existing wiki-based EAM solutions. Based on the experience gained with the use of Hybrid Wikis, new requirements were collected and discussed. We consolidated the findings from the application of Hybrid Wikis by the community partners three times a year into technical reports. These reports were made accessible to the members of the community only. However, the user feedback helped to incrementally improve Hybrid Wikis, that is, according to [He04], to continuously refine the developed research artifact.

Some community members evaluated the capabilities of Hybrid Wikis for a certain period of time and noticed that classical EAM solutions are more suitable for their purposes, for the following reasons (extract from email conversations with these members):

- Hybrid Wikis do not support planning tasks regarding the future EA landscape.
- Classical EAM tools follow a top-down modeling approach, thereby minimizing the risk of data chaos.
- Hybrid Wikis are a research prototype and cannot provide for example maintenance and support in the long-term.

Five participants (i.e., organizations) of the initial workshops left the community. However, in April 2012 the community has grown to 67 members from 48 different organizations. Ten organizations regularly attend the workshops, the remaining rather sporadically, but some of them currently using Hybrid Wikis in research projects in cooperation with the chair (e.g., [Bi12]).

In the 6th workshop in December 2011, we asked (paper-based) the community members⁴ to provide the main reasons why they are using Hybrid Wikis in their enterprises. The participants of the workshop stated (most frequent answers) that

- the information model is flexibly adaptable (i.e., can be created incrementally and does not need to be fixed in advance) (6 of 7),
- Hybrid Wikis are easy to use (i.e., provide a high level of usability and a clean user interfaces) (4 of 7), and
- hybrid searches (cf., Section 3.2.4.1) provide fast access to unstructured and structured contents (4 of 7).

Although these survey results do not represent a well-founded qualitative evaluation, they allow to assume that the adaptability of the structures is the main reason for applying Hybrid Wikis in EAM scenarios. Furthermore, the answers indicate that business users understand the concepts underlying Hybrid Wikis since they adapt structures without additional IT-support facilitated by a web interface that the participants consider to be usable.

In the next sections, we introduce selected case studies⁵ from enterprises participating in the Wiki4EAM community using the following structure:

- Briefly introduce the enterprise.
- Sketch the purpose of application and the perceived benefits.

³<http://www.infoasset.de/wikis/infoasset/download-tricia>; visited on March 10th 2012.

⁴Seven members attended the community meeting in December 2011.

⁵According to [He04], case studies emphasizes research rigor.

- Highlight the structures that emerged.

6.1.1.1. Kassenärztliche Vereinigung Bayerns

“The Bavarian Association of Statutory Health Insurance Physicians (Kassenärztliche Vereinigung Bayerns, abbr. KVB or BASHIP) is a statutory body under public law. It is one of the healthcare system’s self-administration bodies, and its work is under the legal supervision of the Bavarian Ministry of Health. However, it is not a subordinate government agency. The KVB ensures that about 10 million compulsory and voluntary SHI-members in Bavaria can consult a physician or a psychotherapist of their own choice at any time and anywhere in the state. The KVB guarantees high quality of medical care and ensures that every patient can reach a doctor within a reasonable distance. It makes medical care available within and outside consultation hours and continuously works to improve medical services. The KVB represents about 24,000 doctors and psychotherapists and lobbies for preserving and improving the medical system.”⁶

The KVB⁷ participates in the Wiki4EAM community since June 2011. Its main objective is to build an integrated EA repository by means of Hybrid Wikis. In this case, integrated means to collaboratively collect information from three different EA levels, the business level (e.g., business units), the application level (e.g., information systems), and the infrastructure level (e.g., information typically maintained by a *Configuration Management Database* (CMDB), such as infrastructure elements) at a central place. In particular, it is required to make the connections between these different levels visible (e.g., the connection between infrastructure elements and information systems) in order to obtain an integrated, holistic view of the KVB’s EA.

The KVB gained access to Hybrid Wikis through a system installed locally. Before working with Hybrid Wikis, EA information was scattered across the KVB’s technical departments. If available, in many cases such information was independently organized by the departments themselves, for example by using a spreadsheet for product portfolio documentation. In order to gather these information pieces the head enterprise architect of the KVB organized several workshops with representatives of the technical departments in a first step. The main goal of the workshops was to create a common information model in which the different model parts were elaborated and confirmed by the responsible persons of the departments. In this initial workshop phase, Hybrid Wikis served as a basis for discussions about the model’s constituents and its terminology. The workshop organizer (i.e., the head enterprise architect) prefilled the EA wiki with an information model⁸ provided by iteratec GmbH⁹, a German IT consulting company. He mapped this model to types, attribute definitions, and constraints according to the terminology used by iteratec. Additionally, he added some KVB-specific reference data to the wiki before the workshops were carried out.

Based on this data, the workshop participants discussed which attributes to transfer from the department’s own information sources (e.g., spreadsheets) to the wiki, which attributes to define in addition, and which attributes to delete. It was consciously decided to include only

⁶<http://www.kvb.de/fileadmin/kvb/dokumente/UeberUns/KVB-About-us.pdf>; visited on March 3rd 2012.

⁷<http://www.kvb.de>; visited on March 3rd 2012.

⁸<http://www.iteraplan.de/wiki/display/iteraplan/iteratec+Best-Practice-EAM>; visited on April 3rd 2012.

⁹<http://www.iteratec.de>; visited on April 3rd 2012.

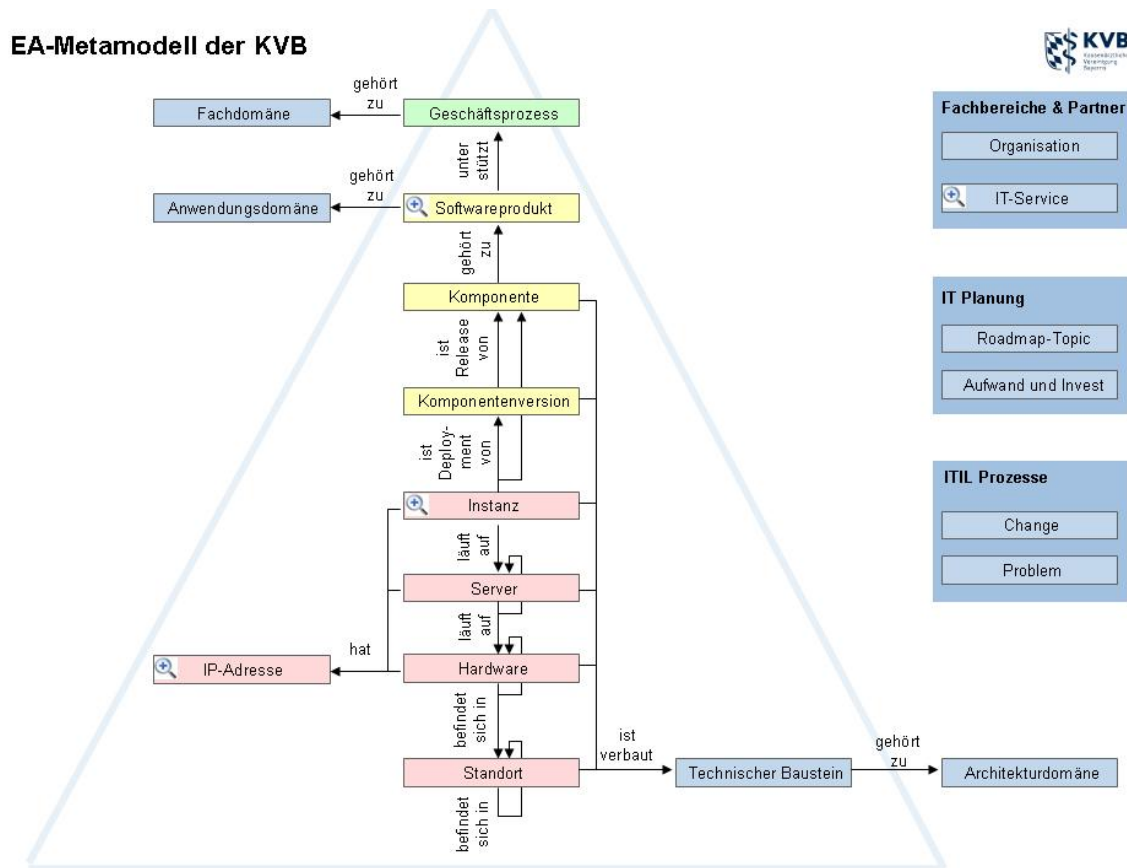


Figure 6.2.: Information model used by the KVB for the management of the EA.

attributes in the wiki relevant for specific EA questions. Furthermore, the iteratec information model was adapted as required by the technical departments. For instance, type names were changed according to the terminology normally used in the departments and types provided by the iteratec model but not required by any department were deleted. The information model created during the workshops is depicted in Figure 6.2.

After the workshops, members of the departments gained access to Hybrid Wikis. Subsequently, they started entering data according to the previously defined information model. All departments obtained edit access to the EA repository (i.e., the EA wiki). For this reason, some persons observed (cf., Section 3.2.4.4) the sets of objects they are responsible for in order to be aware of changes. Even if it is possible with Hybrid Wikis to observe a complete space (e.g., the EA wiki) or individual content objects (e.g., a specific wiki page) some members expressed the need getting informed only in case of objects with a specific type are created or deleted. Additionally, in some cases it was required to specify disjunctive queries. Both requirements are currently not supported by Hybrid Wikis.

Furthermore, an interesting observation was that even if all members were allowed to modify the structures defined in the workshops, attributes and types in the EA wiki were only changed after a short personal consultation with the head architect. This shows that Hybrid Wikis at the KVB are not primarily used to collaboratively develop an information model bottom-up

but rather to maintain structures that were previously discussed with the different stakeholders (e.g., members of the technical departments, head enterprise architect). However, the structures were created and adapted by the users themselves without external help (e.g., IT experts). Since the KVB did not attend the initial Wiki4EAM workshops, the staff was only introduced to Hybrid Wikis by means of a 10-minute screencast available at the official research project website¹⁰ and some slides explaining the core concepts of Hybrid Wikis. This shows that business users (i.e., enterprise architects) can productively work (i.e., in particular structure information) with Hybrid Wikis with little instruction effort.

In addition to the EA repository, the KVB created a second wiki. This wiki serves as a glossary for master data (e.g., specialist doctors, groups of doctors, offices, personal data) and the relationships between this data (e.g., between doctors and offices). Each kind of data is described in an individual wiki page with type tag *glossary entry*. The page's name corresponds to the master data's kind (e.g., personal data). The KVB consciously decided not to use a type tag in this case since for example personal data is not intended to be instantiated in the wiki. The glossary pages are rather used to collaboratively discuss about the meaning of the master data's attributes. The discussion about the attributes takes place in the wiki text (i.e., built-in content). The fact that type tags are only used if it is planned to create concrete instances of that type shows that business users understand the structuring concepts underlying Hybrid Wikis.

6.1.1.2. UniCredit Business Integrated Solutions S.C.p.A. (formerly UniCredit Global Information Services S.C.p.A.)

UniCredit Business Integrated Solutions S.C.p.A. (formerly UniCredit Global Information Services S.C.p.A.) is the global service company, owned by UniCredit¹¹, and born on the 1st of January 2012. It is dedicated to providing services in the sectors of Information and Communication Technology (ICT), Back Office and Middle Office, Real Estate, Security and Procurement. The Company includes about 13.000 people and oversees activities in 11 countries: Austria, Germany, Italy, Poland, Great Britain, Czech Republic, Romania, Slovakia, Hungary, New York and Singapore.

In December 2010, UniCredit Business Integrated Solutions S.C.p.A. (formerly UniCredit Global Information Services S.C.p.A.) started participating in the Wiki4EAM community to deepen the knowledge about EAM. In particular, the unit *Infrastructure Architectures* evaluated whether a previously created infrastructure landscape can be represented in a model built with Hybrid Wikis. In an additional¹² two hours personal lesson with one member of the unit *Infrastructure Architectures*, the core concepts (e.g., wikis, wiki pages, type tags, attributes, constraints) were explained in detail and technical questions about some system functions (e.g., how to define a search, how to embed a search in a wiki page) were clarified. Subsequently, UniCredit Business Integrated Solutions S.C.p.A. (formerly UniCredit Global Information Services S.C.p.A.) gained access to Hybrid Wikis through a system hosted at TU München including some EAM demo data (e.g., business applications, organizational

¹⁰<http://www.matt.hes.in.tum.de/wikis/sebis/hybrid-wiki>; visited on March 10th 2012.

¹¹<https://www.unicreditgroup.eu>; visited on April 19th 2012.

¹²In addition to the initial introduction in the Wiki4EAM workshops.

6. Application and Evaluation

units) and some exemplary visualizations (e.g., a cluster map showing the relation between organizational unit and business applications).

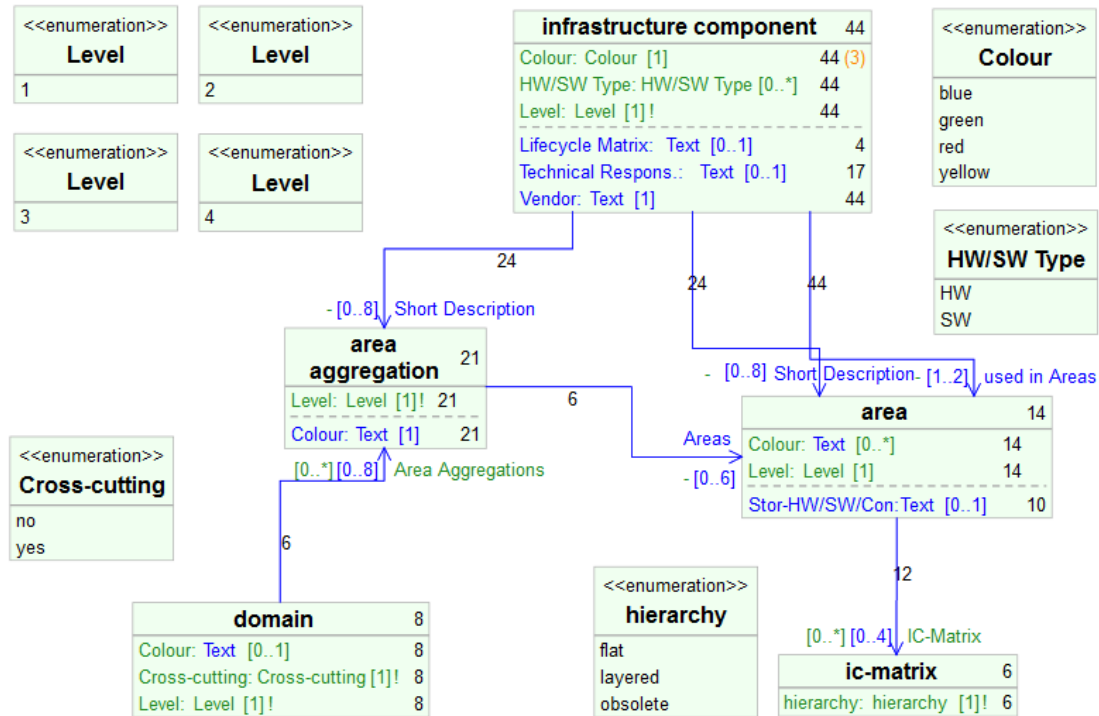


Figure 6.3.: Emergent information model used by UniCredit Business Integrated Solutions S.C.p.A. (formerly UniCredit Global Information Services S.C.p.A.) for the management of infrastructure elements.

In a first step, from December 2010 to December 2011, UniCredit Business Integrated Solutions S.C.p.A. (formerly UniCredit Global Information Services S.C.p.A.) prototypically modeled the architectural elements of the infrastructure unit in a specific wiki (i.e., separated from the wiki containing the EAM demo data). The emerged data model representing the infrastructure architecture (i.e., infrastructure landscape) is depicted in Figure 6.3. The concepts of this model represent infrastructure elements on a certain level of detail each. The model consists of 4 levels. On the lowest level (i.e., the most fine-grained), an *Infrastructure Component* (IC) (level 4) can either be software (e.g., Windows 2000 Server) or hardware (e.g., Itanium). An area (level 3) is the logical unit for a set of ICs (e.g., a set of Windows software components), an area aggregation (level 2) is the composition of areas (e.g., in a mainframe), and a domain (level 1) composes a set of area aggregations (e.g., a database and middleware composition).

The wiki's home page is used as a dashboard. It shows all available domains, areas, and area aggregations as custom embedded lists (cf., Section 3.4.2). ICs were also shown as an embedded list on that dashboard page in the beginning. However, the list of all ICs became too long over time. Therefore, the list was replaced by a link to a separate wiki page showing the ICs as a custom embedded table (cf., Section 3.4.2). It was consciously decided not to link to the type table view (cf., Section 3.4.1.5) in this case since only a subset of the available attributes was required to be shown in this table. Furthermore, the dashboard indicates all obsolete

infrastructure components by using a custom embedded table view. Obsolescence is indicated by ICs having a *color* attribute with value *red*. In order to ensure that each IC provides a color a multiplicity constraint (exactly-one) is specified. Additionally, an enumeration constraint is defined having the values *blue*, *green*, *red*, and *yellow*. The meaning of each color is given in the attribute definition's description field. For instance, the enum value *red* is explained as *discontinue use: No further investments; plans for migration or deinstallation exist*.

Finally, the dashboard shows the relationships between

- domains and area aggregations,
- area aggregations and areas, and
- areas and infrastructure components

as custom embedded tables. Each table shows only a subset of the available attributes. Besides the tabular representation, these relationships are visualized as a cluster map each. Therefore, embedded tables and graphical visualizations are means to filter for specific information according to the business user's needs.

In April 2011, UniCredit Business Integrated Solutions S.C.p.A. (formerly UniCredit Global Information Services S.C.p.A.) draws the following lessons learned from their experiences with Hybrid Wikis in a Wiki4EAM community workshop:

- Absolute beginners are not able to start without any introduction, that is, a short lesson (about 2 hours) or paper is mandatory.
- Creating and adapting information models is possible without any further help from IT experts after an introduction.
- Starting with data is simpler than creating a meta model first.
- Having a previously developed model (as in this case a model of infrastructure landscape) in mind is helpful since it provides at least a basic frame for structuring and restructuring in Hybrid Wikis is time consuming.
- Gradually adding attributes to wiki pages works well and creates beneficial structures.
- Inverse links shown in the wiki pages facilitate navigation and structuring.
- Configuring visualizations manually is cumbersome.
- Demo data and exemplary configurations of visualizations are helpful.

6.1.1.3. Miles Group GmbH

Miles Group GmbH¹³, an Austrian company, was founded in 2002 as property developer for residential building. In 2007, the company was restructured and focused on commercial property as for example hotels and retail parks. Furthermore, the company is running a consulting division carrying out the redevelopment of already built building according to legal and technical specifications. Up to 2010 a division IT consulting and computer center operations

¹³<http://www.miles-group.at>; visited on April 19th 2012.

6. Application and Evaluation

was run, which is nowadays in its own company. This division is primarily concerned with IT project management offices and operates infrastructure in a computer center for special applications of costumers.

Hybrid Wikis are mainly used to manage outgoing invoices for example to observe the payment status. The invoices are kept in typed wiki pages referencing the invoice document (Word and PDF) originally sent to the customer by using a link attribute. The documents are uploaded via the SMB protocol-based file share supported by the Tricia platform. Each invoice page indicates the customer it belongs to by using a structured attribute. In case of large customers, the attribute is a link to a typed page representing the customer, in case of small ones, a simple string value is used. String values are transformed to links while the customer grows. Additionally, several prominent properties of an invoice are denoted as attributes, such as creation date, state, or invoice amount (cf., Figure 6.4). Finally, the invoice indicates the company's division (e.g., information technology, project management).

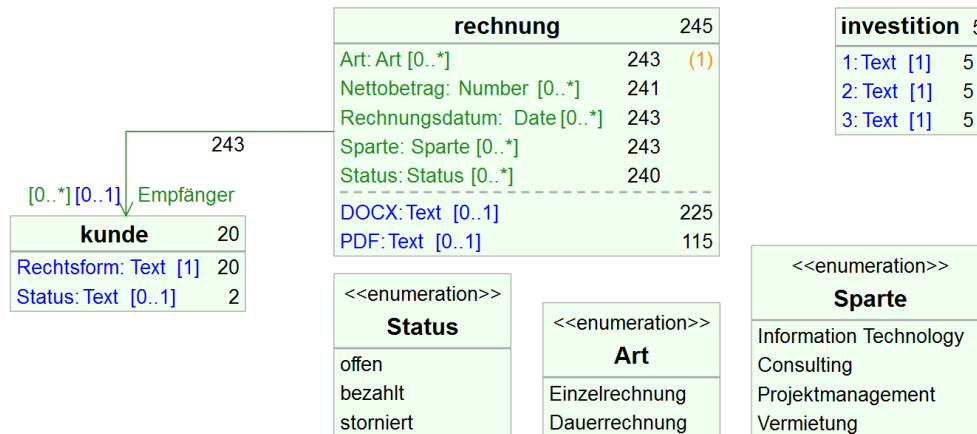


Figure 6.4.: Emergent information model used by Miles Group GmbH for the management of invoices.

Based on this simple data model, Miles Group GmbH created a dashboard page by using custom embedded views. For example, an embedded list view (cf., Section 3.4.2) helps to detect all unpaid invoices (i.e., invoices with status *open*) or a list of all invoice numbers is generated in order to ensure the consistency of all numbers across divisions. These lists make it easy to keep the accountants department up to date and allow them to access live information about invoices. Finally, Hybrid Wikis support users in finding specific invoices by using Tricia's generic search module¹⁴, for instance by entering the invoice number, date, or customer name. This is necessary for example to answer incoming questions from customers regarding a specific invoice or to quickly navigate within the wiki in order to update the invoice status or add additional structured information, such as payment dates.

According to plans, the usage is incrementally broadened to interlink invoices with properties or to widen the use of customer pages not only with invoices but also with projects and other

¹⁴Hybrid Wikis extend Tricia's generic full text search by also taking into account attributes and types (cf., Section 3.2.4.1).

information. Currently about 250 invoices are kept in the system, with around 500 documents and around 20 key customers.

6.1.2. InfoAsset AG

InfoAsset AG¹⁵ is developing web-based information management solutions since 1999. Since 2010 the company focuses on a scalable product-based business by providing Tricia as Enterprise 2.0 platform for commercial use. Among other things (e.g., CRM, license management) InfoAsset AG uses Hybrid Wikis for its issue management, in particular for the management of issues regarding the Tricia product.

Customers can send feedback¹⁶ to the official InfoAsset server (i.e., a Tricia server having the Hybrid Wikis plugin activated) on issues (e.g., change requests, error messages) regarding their installations. For each new emerging issue a structured wiki page is automatically created typed as *issue*, providing an attribute *status* with value *unprocessed*, and having some additional attributes, such as *submission date*, *installation id* (i.e., the URL of the installation), *user email*. The Tricia developer dashboard shows a list of all unprocessed issues by using a custom embedded view filtering types and attributes accordingly. When a new entry appears in this list (i.e., the underlying embedded query yields a new search hit) the developers are informed by means of a new entry in an RSS feed. A responsible person (i.e., the issue manager) processes these new entries and categorizes them by assigning additional type tags. Currently the three additional type tags *bug*, *feature request*, and *refactoring* are primarily used. Furthermore, a person is assigned responsible for processing this issue by setting the attribute *assigned to* to the respective name or directly linking to the developer's profile page. Additionally, the *status* is set to *open* in order to remove this issue from the dashboard list view. Further information is given about for example the estimated effort, the priority, and the related customer project. The responsible developer is in turn informed by an RSS-Feed and the issue appears in her personal dashboard, showing all issues assigned to her in descending order by priority.

An extract of the data model used for Tricia's issue management is shown in Figure 6.5. The depicted model elements almost completely emerged bottom-up by the management of structured wiki pages with attributes and types. The following constraints were defined top-down after the structures had become more stable, as for example:

- Enumeration constraints for the attributes *status* (e.g., *open*, *closed*, *solved*), *priority* (e.g., *high*, *low*, *medium*), and *effort* (e.g., *low*, *medium*, *high*). The enumeration values were plain strings in the beginning.
- *exactly-one* multiplicity constraint for the attributes *status*, *effort*, *priority*.
- *exactly-one* multiplicity constraint for the link attribute *assigned to*.

Since the attribute *assigned to* is declared as required for unprocessed user feedback a validation message is shown. This additionally encourages the issue manager to specify a responsible developer.

¹⁵<http://www.infoasset.de>; visited on April 19th 2012.

¹⁶Feedback can only be send if the installation uses a specific plugin and is allowed to access the Internet.

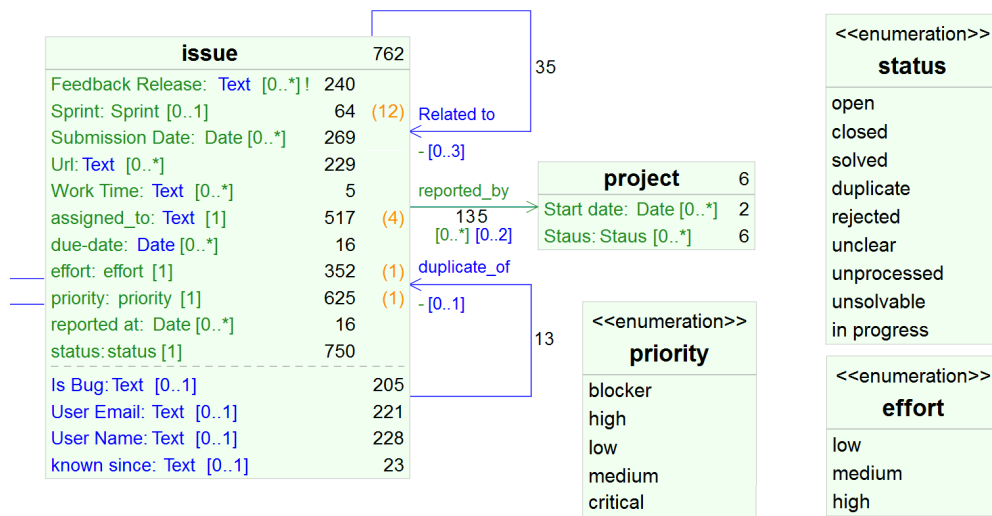


Figure 6.5.: Emergent information model used by InfoAsset AG for the management of issues.

6.1.3. TU München (sebis)

The author's chair *Software Engineering for Business Information Systems* (sebis)¹⁷, established in 2002, is one of the 21 chairs of the Informatics Faculty at Technische Universität München. sebis contributes to the informatics and business informatics education at the PhD, master, and bachelor level. In March 2012, the chair employed 11 people.

Among many other things (courses, publications, tasks, meeting minutes, paper writing, infrastructure and application documentation, contacts, etc.) sebis uses Hybrid Wikis for the management of course exercises. For instance, in the summer term 2011 Hybrid Wikis were used to manage a course with about 500 students and 23 weekly exercises held by 12 different tutors.

The information model used for the management of this course is depicted in Figure 6.6. The course's exercises are represented by structured wiki pages (typed with *exercise*) with some attributes, such as *room*, *time*, *tutor*, and *participants*. These pages can be read and edited by the course organizers and tutors. *Tutor* attributes contain link values referencing the tutor's profile pages, *participants* are links to pages typed as *student*. Student pages have attributes such as *first name*, *last name*, and *email*. Furthermore, a *partner* attribute indicates that two students are part of the same team. These pages can be edited by the tutors and additionally be read by the corresponding student.

The tutors entered the grades of the students' weekly homework as attributes in the student pages (e.g., *credits exercise 1*, *credits exercise 2*). Additionally, the tutors provided some correction notes and explanations in the pages' built-in content. Some students observed their pages in order to get notified (e.g., by email) in case of changes (e.g., new entered grades). Some students used comments¹⁸ to discuss about their homeworks and grades with their tutors. In such cases, even the tutors registered to receive change notifications.

¹⁷<http://www.matthes.in.tum.de>; visited on April 19th 2012.

¹⁸Tricia allows to comment pages even if the read access is restricted to some persons.

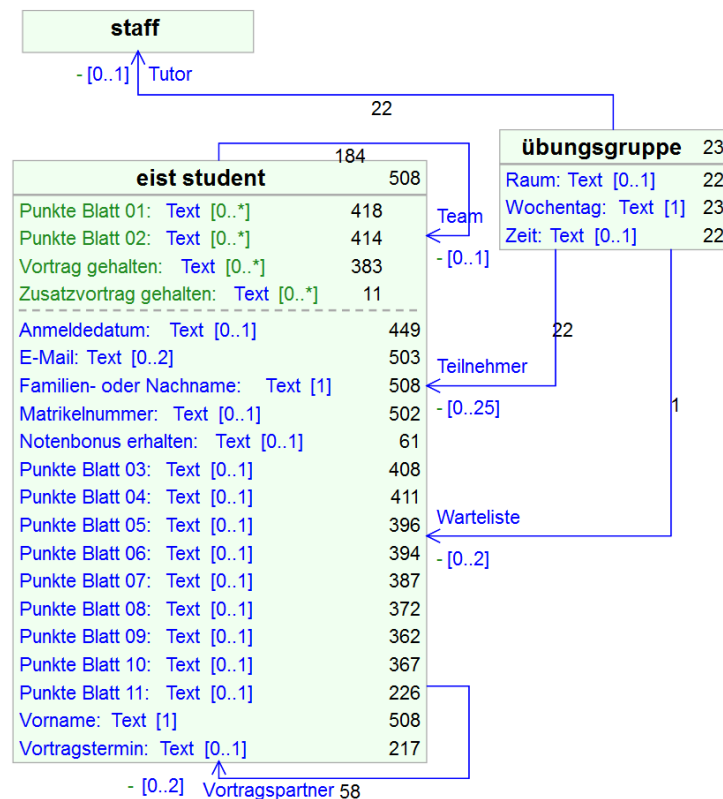


Figure 6.6.: Emergent information model used by sebis to manage students and exercises of a course.

Some tutors created custom embedded table views within the exercise pages (they were responsible for) showing the participating student of an exercise (i.e., the exercise represented by that page). Furthermore, some defined additional columns in the tables (i.e., by means of projection) knowing that there will be further similar attributes (e.g., *credits exercise 10*) in the future. By means of these table views it was convenient for the tutors to manage their relevant information by entering the grades like in a spreadsheet.

The initial structures (student pages and exercise pages) were defined by the exercise organizer. The students and exercise assignments were imported based on a spreadsheet exported from a tool the students use to schedule their semester. Furthermore, an attribute definition *credits exercise 1* was created for type *student*. The tutors followed this pattern by entering all grades as structured attributes (e.g., *credits exercise 10*), but no further attribute definitions regarding the credits were specified. Only one additional attribute definition was created indicating that a presentation was held by the students as part of the exercise.

The exercise organizers used the student's type table view to check if the tutors entered the grades on time. Furthermore, the structured search helped them to get an overview of the student's performance per exercise (e.g., in order to identify high potentials).

6.1.4. Pixida GmbH

Pixida GmbH¹⁹ is a German engineering consultancy. It supports customers from different engineering disciplines in project management and execution. Customers for example are Audi²⁰, BMW Group²¹, and Schreiner LogiData²². The core disciplines of Pixida GmbH are infotainment, telematics, mobility, and IT consulting²³.

In [Gr12] the author, employed at Pixida GmbH, examines how the business processes of Pixida GmbH can be supported by Hybrid Wikis. In particular, he examines how business process can be documented, how quality management of business processes can be supported, and how business processes can be executed by means of Hybrid Wikis. After a short theoretical discussion about the potentials of Hybrid Wikis in supporting business processes he describes the experiences made in applying them at Pixida GmbH in a case study.

Pixida GmbH accessed Hybrid Wikis through a system installed locally. The business processes of each organizational unit (e.g., recruiting, quality management) were documented by using different wikis. The staff gained read access to all wikis, members of a organizational unit gained write access to their corresponding wiki (i.e., the wiki representing the unit the members belong to). The author of [Gr12] initially created this basic wiki structure, persons, groups, and access control lists. In a first step, the business processes (type tag *process*) of the human resource department were collaboratively documented, for example *leave requests*, *basic equipment*, and *education and training*. A process provides for example the attributes *status*, *responsible person*, *participants*, *partner*. These attributes were explicitly defined (i.e., by means of attribute definitions) as part of the type *process*.

Members of the human resources department created new business processes by cloning a wiki page (with type tag *process*) that serves as a template. This template page was initially created by the author of [Gr12]. The wiki text of the template includes hints about the structures' meaning, guidelines for business process documentation, and some example data. Even if this basic structure was predefined users adapted process pages cloned from this template according to their needs. For instance, users introduced new attributes or they embedded images in the built-in content showing the current business process represented in *Business Process Model and Notation* (BPMN) 2.0 for purpose of illustration. Additionally, some processes were described and discussed (e.g., individual process steps) within the wiki text.

Parts of these process descriptions and some of the attributes should only be visible for the members of the wiki the process is documented in (i.e., for the members of the organizational unit) for purpose of readability for staff of other units. Since Hybrid Wikis currently support access control neither for parts of the built-in content nor for individual attributes, the members moved these elements (i.e., content parts and attributes) to additionally created wiki pages. That is, for each documented process they created an additional internal process page (with type tags *process* and *internal*) containing the content parts and attributes only relevant for the members of the unit. In order to preserve the connection between both pages

¹⁹<http://www.pixida.de>; visited on April 19th 2012.

²⁰<http://www.audi.de>; visited on April 11th 2012.

²¹<http://www.bmwgroup.com>; visited on April 11th 2012.

²²<http://www.schreiner-logidata.com>; visited on April 11th 2012.

²³<http://www.pixida.de/4/About%20us.html>; visited on April 11th 2012.

Wikis » Wiki-Personalabteilung » Type Tags

View Hierarchical Details Versions Type Tags

All Type Tags in Wiki-Personalabteilung

Type Tag	Type Definition
intern (14)	Prozessaufnahme-Status (14) Verantwortlich (13) Mitwirkend (9) Partner (extern) (5) Informiert (3)
organisationsbereich (1)	Prozesse (1)
partner (extern) (5)	Adresse (5):Text Ansprechpartner (5):Text Fax-Nummer (3):Text Homepage (1):Text Telefonnummer (5):Text eMail (1):Text
personalabteilung (1)	Prozesse (1):Link
personalprozess (28)	Informiert (5):Link Mitwirkend (18):Link Partner (extern) (9):Link Prozessaufnahme-Status (28):Enumeration Verantwortlich (26):Link
visitenkarten formular (9)	Auftragsdatum Pixida-Visitenkarte (6):Date Auftragsdatum Projekt-Visitenkarte (4):Text Druckdaten Pixida-Visitenkarte (0):Link Druckdaten Projekt-Visitenkarte (0):Link Status Pixida-Visitenkarten (9):Enumeration Status Projekt-Visitenkarten (9):Enumeration Mitarbeiter (8)

Figure 6.7.: Emergent information structures used by Pixida GmbH in the human resource department for business process documentation according to [Gr12].

the original process page is referenced in the wiki text of the internal page. In these ways, wiki page readers are not disturbed by irrelevant information. However, the members of the unit (in particular the authors of the process pages) were burdened with additional information management and navigation effort.

The structures that emerged in the human resource department are depicted in Figure 6.7. Pixida GmbH additionally uses the types *customer*, *meeting*, and *event* to manage further business activities in other wikis. The customer's address attribute uses default values in order to provide placeholders facilitating the entry of for example *street*, *city*, and *country* (cf., Figure 6.8). The system's home page shows the processes, events, and meetings of all wikis by using customer embedded lists. The human resources wiki's home page shows the unit's business processes as a custom embedded table and as a cluster map (cf., Section 6.1.1). Additionally, the author of [Gr12] prepared some custom embedded table views for multiple purposes. For example, a quality manager can use a table view to generate reports showing all upcoming internal audits or to display general improvement suggestions from employees.

Finally, some users observed (cf., Section 3.2.4.4) the human resource wiki in order to get informed in case of newly created business processes or changes to existing ones.




Types: kunde	
Adresse 	<Straße und Hausnummer> <PLZ> <Ort> <Staat>
Ansprechpartner 	
Fax-Nummer 	
Homepage 	
Telefonnummer 	

Figure 6.8.: Default values used as placeholders facilitating the data entry of customer addresses according to [Gr12].

In [Gr12] the author confirmed that constraints helped to avoid inconsistencies in managing structured information and types to reduce maintaining efforts by allowing the reuse of existing structures. Furthermore, he pointed out that the staff learned information structuring with little introduction effort. The adaptability of structures in Hybrid Wikis empowered Pixida GmbH to incrementally develop an information model bottom-up without the need to have a complete model in mind in advance. However, this way increased coordination and communication efforts regarding the structures were required in the beginning. Additionally, some users expressed a need to define type-subtype relationships in order to inherit attribute definitions and constraints, for example in order to share the definition of addresses between the types *customer* and *partner*.

6.1.5. Summary

In this section, we presented the experiences gained in applying Hybrid Wikis in business contexts by introducing selected case studies. Tables 6.1, 6.2, 6.3, and 6.4 summarize the different purposes Hybrid Wikis are used for in six enterprises by listing the types primarily applied. The productive use of Hybrid Wikis showed that business users indeed create and adapt information structures according to changing demands without the need to involve IT experts. In most cases, only little introduction effort was required to familiarize users with the concepts and mechanisms of Hybrid Wikis.

In two cases Hybrid Wikis were well known to the users. *sebis* is the author's chair and *InfoAsset AG* a company commercializing the software Hybrid Wikis are based on. We included these cases not to validate our approach but rather to illustrate the broad range of applicability. In both cases, we presented only an extract of the types that are used. Particularly interesting is that most business information management activities in these two organizations are supported by Hybrid Wikis without the need for special purpose applications (e.g., a CRM tool or an issue tracker). This has the advantage that most business critical information is integrated in a central knowledge repository, can be linked, and searched. It would be interesting to

Table 6.1.: Hybrid Wikis used for enterprise architecture management.

Enterprise, EA layer	Business & Organization	Application & Information	Infrastructure & Data
KVB	business process	component, software product	location, hardware, server
UniCredit Group			area, area aggregation, domain, infrastructure component
sebis		application, deployed application	host, vm-host
Pixida GmbH	organizational unit, process		

Table 6.2.: Hybrid Wikis applied in enterprises supporting customer-relationship-management, human resources, events, meetings, and accounting.

Enterprise, Purpose	CRM	HR	Events	Meeting	Accounting
Miles Group GmbH	customer				investment, invoice
sebis	company, contact	staff	event, workshop, presentation	discussion, meeting, telco	fond, payment
Pixida GmbH	customer, partner		event	meeting	
InfoAsset AG	opportunity, support	person, staff	event	protocol	invoice

examine at which point of time and for what reasons enterprises introduce special purpose tools for information management that is currently supported by Hybrid Wikis. However, this is left for future research.

6.2. Experimental examination of models created with Hybrid Wikis

In [Ra11] we conducted a laboratory experiment with 11 students at Technische Universität München evaluating the quality of models created with Hybrid Wikis compared to models built with a UML tool. In particular, the author of [Ra11] analyzed the models created during the experiment regarding their structural complexity, completeness, and validity.

Table 6.3.: Hybrid Wikis applied in enterprises supporting the management of projects, tasks, configurations, and templates.

Enterprise, Purpose	Project management	Task management	Configuration management	Templates
sebis	project, research project, student project	issue, responsibility, team task		form
Pixida GmbH				business card
InfoAsset AG	project	task	bug, feature request, issue, refactoring, release, sprint	

Table 6.4.: Hybrid Wikis applied in enterprises supporting personal information management, licenses, lectures, and publications.

Enterprise, Purpose	Personal Information Management	Licenses	Lectures	Publications
sebis	idea, note, thought, todo		course, exercise, lab course, seminar, staff, student	article, bachelor thesis, book, conference, journal, master thesis, publication, thesis
InfoAsset AG	idea	host, license		

According to [Ra11], the research questions underlying that thesis are

- Which experimental setting and design are appropriate to evaluate the UML models generated by Hybrid Wikis?
- Which types of measures can be used to determine the quality of models?
- Is the structured part of data entered in Hybrid Wikis as good as the structure of UML models?

The author of [Ra11] identified a set of metrics necessary to determine the models' pragmatic quality (complexity) and semantic quality (validity, completeness) in order to answer these questions. For instance, complexity is determined by the number of modeled classes,

validity by the number of wrong classes, and completeness by the number of missing classes (cf., Figure 6.9).

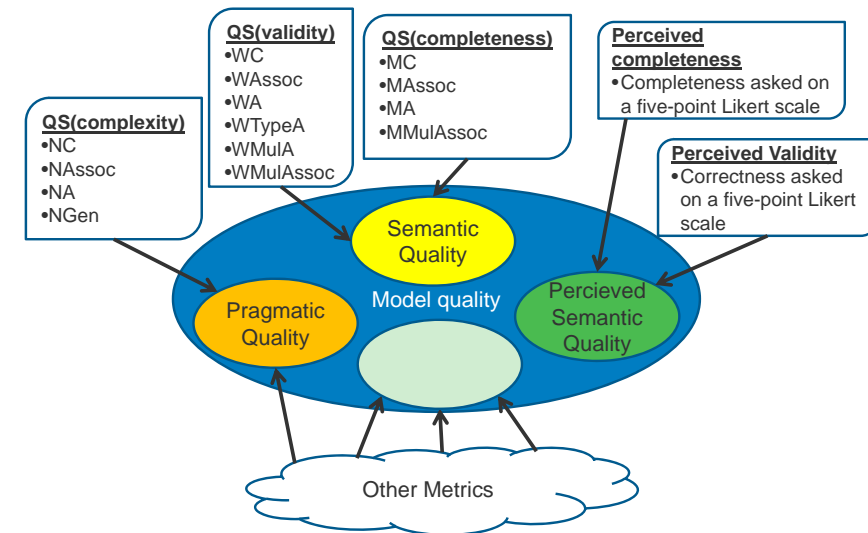


Figure 6.9.: Metric used to experimentally evaluate the quality of UML models and models created with Hybrid Wikis according to [GP01, Kr95, Ra11].

The participating students were randomly divided into two groups (A and B). In the experiment's first phase, group A was introduced to Hybrid Wikis and group B to UMLet²⁴ (a lightweight tool supporting UML modeling) using a video tutorial and slides. Subsequently, the students received an exercise consisting of a textual description from the domain of project management. The students' job was to transfer this textual description to the modeling concepts provided by the tool used, in UMLet by creating classes, properties, and associations, in Hybrid Wikis by creating wiki pages with type tags, attributes, and constraints. In the second phase, the participants executed the same exercise but using the other tool (i.e., A uses UMLet and B Hybrid Wikis), also getting introduced in the tools by video and slides before. Subsequently, the identified metrics were applied to the models resulting from the experiment in order to determine the best models from both tools in terms of pragmatic quality and semantic quality. Finally, in order to determine the perceived semantic quality (i.e., perceived completeness and perceived validity) the participants were asked open and closed (Likert scale) questions about these two models in a paper-based questionnaire.

The answers given in the questionnaire indicate that the participants perceived the designated best UML model to be more complete than the best model from Hybrid Wikis. Nevertheless, the students preferred to work with Hybrid Wikis instead of UMLet: "Four prefer to work with Hybrid Wiki, four are neutral, and two prefer to work with UML". Additionally, answers from the open questions show that Hybrid Wikis are easy to use and understand. However, the students expressed a need for creating type hierarchies.

Furthermore, a statistical analyses of the results (i.e., all models resulting from the experiment) based on the metrics shows that models created with Hybrid Wikis

²⁴<http://www.umlet.com>; visited on March 2nd 2012.

- are slightly less complex (but not statistically significant),
- are slightly more complete (but not statistically significant), and
- have less wrong constructs (with a probability of 95 percent)

than UML models.

Finally, the author draws the following lessons learned, in particular she claims to

- select a larger number of representative subjects (also participants without modeling skills) in future experiments in order to obtain more statistically significant results,
- emphasize the difference between conventional tags and type tags since it was not obvious for one student,
- implement type hierarchies in order to inherit attribute definitions, and
- show backlinks in the type table view in order to manage inverse roles (e.g., names, multiplicities) for attribute definitions.

From the fact that the participants never worked with Hybrid Wikis before and only were introduced by a fifteen minute video the author concludes that Hybrid Wikis empower users to model real-world scenarios (e.g., project management) with little introduction effort [Ra11].

6.3. Evaluating and visualizing the structural evolution of Hybrid Wikis

In [Tr12] the author examines how the structural evolution of Hybrid Wikis can be analyzed. In the first phase of that thesis, he developed a data structure that allows to query (based on XPath expressions) the version history of content objects, types, attribute definitions, and constraints by aggregating change events. This data structure allows to efficiently find out for example how many and which attributes an individual wiki page has at a certain point of time within its life-cycle. Based on this data structure, he prototypically developed two graphical visualization integrated in Hybrid Wikis.

- A chart showing the structural evolution of a wiki page (cf., Figure 6.10). By default, the number of assigned attributes, types, and tags are drawn over time. For each change event further information is given (e.g., listing removed attributes or assigned types).
- A chart showing the evolution of a type. By default, the number of attribute definitions and number of instances are drawn over time. For each change event further information is given (e.g., listing added attribute definition or deleted instance).

Even if these charts are simple means to visualize quantitative data, the underlying data structure and query language give users the possibility to analyze the structures of Hybrid Wikis according to their needs. Users can define custom queries and visualize the query results graphically. This way, users are empowered to identify for example correlations in structures or typical structuring pattern.

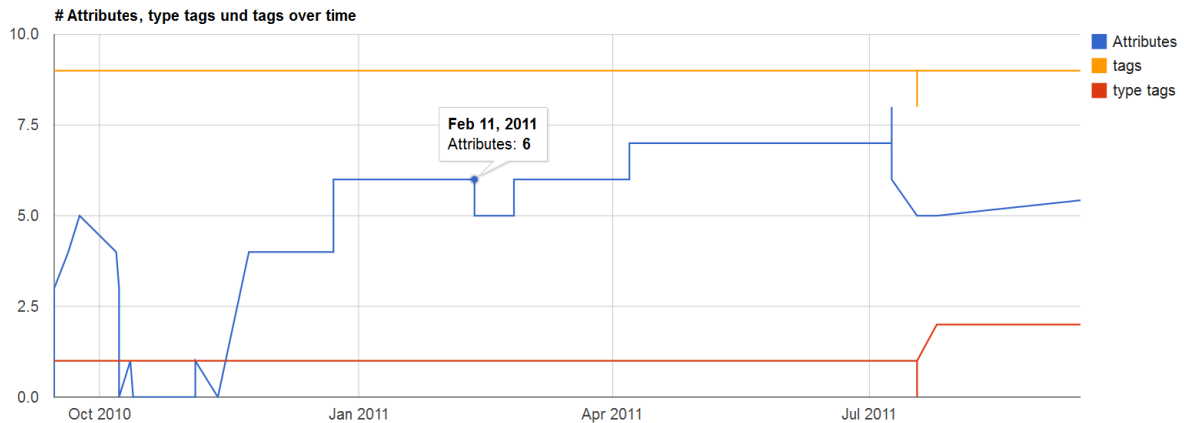


Figure 6.10.: Visualizing the structural evolution of a wiki page with attributes, types, and tags according to [Tr12].

From a research perspective, the approach is a generic means that can be used to confirm or reject hypothesis regarding emergent structures. For instance, it would be possible to examine the hypothesis *Attributes emerge before attribute definitions* by analyzing the structures of different installations from our industry partners. In [Tr12] the author analyzes the public web sites (i.e., wiki pages) of our chair's wiki²⁵ by means of his solution. Subsequently, he discusses and interprets the results of this analysis. However, since the number of public web sites is limited in this wiki analyzing the contents of other application scenarios would be more interesting, but this is left for future research work.

²⁵<http://www.matthes.in.tum.de>; visited on March 10th 2012.

This chapter concludes the thesis by summarizing the results in Section 7.1. In Section 7.2, we provide an outlook on future research based on Hybrid Wikis.

7.1. Summary

In this thesis we presented Hybrid Wikis, an innovative approach to support information structuring in Enterprise 2.0 platforms. Hybrid Wikis empower users to incrementally and collaboratively structure linked information, as well as adapt emergent structures according to changing business demands - both without the need to involve IT experts and without requiring training in conceptual modeling. In the following, we briefly summarize the findings of this thesis and explain how they respond to the research questions outlined in Section 1.2.

First, in Chapter 2 we presented the general ideas underlying Hybrid Wikis by briefly introducing the core structuring concepts (attributes, types, constraints) and exemplifying user interactions based on these concepts. Then, we introduced three principles that guided both the design and development of Hybrid Wikis.

In Chapter 3, we designed a conceptual model that supports information structuring in Enterprise 2.0 platforms. We explained the model's concepts in detail and explained how a schema can be derived from structured content objects with attributes and type assignments. We also showed how this implicit schema can be made explicit with types, attribute definitions, and constraints. We discussed the integration of these concepts in current Enterprise 2.0 platforms and the resulting impacts on typical Enterprise 2.0 services. We explained how the structuring of information can be facilitated by providing suggestions, autocompletion, and objectification mechanisms. We showed how (explicitly defined) information structures can be adapted and harmonized by proposing transitions and consolidation techniques. For the purpose of illustration, we presented a set of built-in views of Hybrid Wikis users can interact with. We

also introduced custom views that allow for showing business critical information according to user-specific needs. The findings from this chapter addressed our research questions 2 and 3.

Chapter 4 responded to research question 4 by explaining how Hybrid Wikis are implemented based on the existing Enterprise 2.0 platform Tricia. Firstly, we introduced Tricia's system architecture and its interfaces. Secondly, we explained how Hybrid Wikis are integrated in Tricia by presenting the developed data model and the extended architecture, as well as by discussing important design decisions. We explained how suggestions can be calculated efficiently. Furthermore, by means of screenshots we illustrated how users can adapt structures by using transitions and consolidation techniques. We described import and export interfaces that allow the exchange of information models and structured content. Lastly, we introduced Hybrid Wikis as means to integrate information scattered across different application repositories within enterprises.

In Chapter 5, we compared Hybrid Wikis to 28 web-based tools or prototypes described in scientific literature. We briefly introduced these approaches and highlighted commonalities and differences. Additionally, we presented findings from a survey on integrated commercial and open source Enterprise 2.0 applications. This chapter addressed research question 5.

The applicability of Hybrid Wikis is demonstrated in Chapter 6. We listed feedback from a community of enterprise architects from 25 large German enterprises (industry and the public sector) shared with us during seven workshops conducted from December 2010 to March 2012. We evaluated Hybrid Wikis by means of case studies in six organizations from different domains, in particular we presented the information structures that emerged by using UML-like diagrams. In a controlled experiment with students at Technische Universität München, we examined the quality of structures created with Hybrid Wikis. The results show that with a probability of 95 percent models created with a UML tool have more wrong constructs than models created with Hybrid Wikis [Ra11]. Finally, we presented a tool that allows to quantitatively analyze structured content in Hybrid Wikis. The chapter responded to research question 6.

By responding to research questions 2-6 we also responded to this thesis' overall research question: We showed that facilitating emergent and adaptive information structures in Enterprise 2.0 platforms is possible.

7.2. Outlook

Hybrid Wikis are an innovation in the field of Enterprise 2.0. However, in previous chapters, we left some questions unanswered and identified potentials for further research. Subsequently, we sketch how these open questions could be addressed and discuss potential research topics based on Hybrid Wikis.

7.2.1. Type hierarchies

Type hierarchies are important means in the area of conceptual modeling [ET11]. The application and evaluation of Hybrid Wikis (cf., Chapter 6) showed that type hierarchies are frequently missing. Even if inheritance can be simulated in Hybrid Wikis by means of multi-

ple tags per object and resulting attribute suggestions, attribute definitions with constraints are currently not inherited. Since attribute definitions have to be maintained in multiple types, this might lead to increased efforts and inconsistencies in the explicitly defined schema. In [MNS12] we discussed how type hierarchies can be detected based on tagged resources. We identified so-called subsumption relationships between tags. Simply put, a tag *project* subsumes a tag *research project* if the set of resources tagged with *research project* is a proper subset of the set of resources tagged with *project*. That is, *project* can be considered to be more general than *research project*. Based on this relationship, in a prototype we automatically assign a (type) tag *project* if a (type) tag *research project* (i.e., the more specific tag) is assigned to a resource - in order to preserve derived hierarchies of (type) tags. This mechanism could also be applied to types in Hybrid Wikis. Attribute definitions could then be inherited by means of an implicitly defined type hierarchy. Alternatively, type relations could be explicitly defined per type. For example, for each type could be defined on which other types it depends on. However, it would be interesting to examine how inheritance can be supported by Hybrid Wikis in both cases (i.e., implicit or explicit type relations), especially having in mind that business users still should be able to understand their concepts and user interface.

7.2.2. Empirical evaluation of the use of Hybrid Wikis

In Chapter 5, we compared Hybrid Wikis to several tools and prototypes by discussing their similarities and differences. In future research, the structures created with these tools and Hybrid Wikis could be analyzed (e.g., regarding the quality of the structures) by means of for example a controlled experiment, similar to the comparison with UML models introduced in Section 6.2, or by analyzing structures that emerged from the application of those tools and Hybrid Wikis in different enterprise contexts.

In Section 6.3, we introduced a tool that allows to quantitatively analyze the structures of Hybrid Wikis. This tool provides the basis for hypothesis-based research. For instance, application data obtained from different installations could be analyzed in order to confirm or reject previously defined hypothesis regarding structures (e.g., in order to identify patterns in the users' structuring activities). The results of such an analysis could help to further improve Hybrid Wikis. For instance, if it turned out that on wiki pages with types and attributes full text is rarely used structured pages could be displayed differently by default (e.g., hide the full text by default).

Furthermore, the case studies presented in Section 6.1 could be broadened. It would be interesting to see whether information model patterns emerge in specific application contexts. Or to find out whether Hybrid Wikis are particularly suitable for specific application domains. In case of enterprises that use Hybrid Wikis for multiple purposes (e.g., projects, EAM, CRM, meetings), it could be analyzed when and for what reasons special purpose tools are being introduced for information management activities that are currently supported by Hybrid Wikis. Also, feedback from Hybrid Wikis' users could be collected on questionnaires or in interviews.

Some concepts and mechanisms provided by Hybrid Wikis have not yet been applied in practice and thus could not yet be evaluated, such as structured links, record and hypertext values,

structuring of other content objects than wiki pages (e.g., persons, blog posts, files), or different thresholds for suggestions. For example, the introduction of structured links could reduce user acceptance due to increasing conceptual expressivity. This could be investigated in future research.

7.2.3. Towards user-adaptive information systems

Hybrid Wikis empower users to incrementally and collaboratively create and adapt data models without needing to involve IT-experts. An interesting research topic would be to examine whether it is possible to develop complete information systems, in terms of data, process, and user interface modeling by following the paradigm of user-adaptability. In particular, it could be examined whether it is possible to implement a user-centered and model-based information system environment that supports both the execution and user-driven evolution of information systems by means of domain-specific modeling languages. It could be tested if information systems built in this environment can be adapted more rapidly and with fewer errors to unforeseen changes in business requirements when a significant number of the necessary system changes can be implemented by business users themselves and if it is even possible that changes to these systems can be applied by users without the involvement of software engineers.

APPENDIX A

Appendix

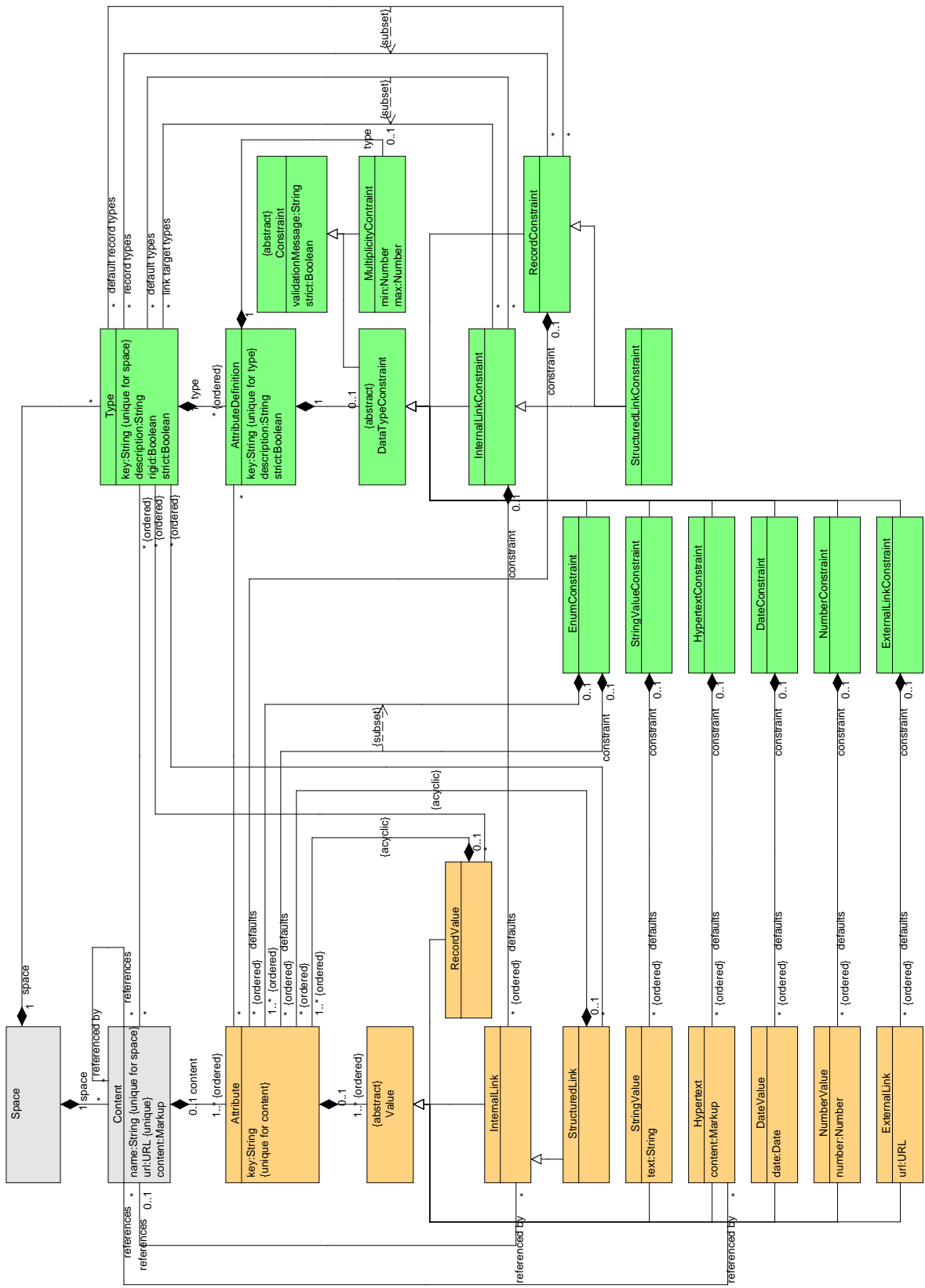


Figure A.1.: The conceptual model of Hybrid Wikis.

- [AA05] Aumueller, D.; Auer, S.: *Towards a Semantic Wiki Experience - Desktop Integration and Interactivity in WikSAR*. In *Proceedings of 1st Workshop on The Semantic Desktop - Next Generation Personal Information Management and Collaboration Infrastructure, Galway, Ireland, Nov. 6th*. November 2005.
- [AC10] Antin, J.; Cheshire, C.: *Readers are not free-riders: reading as a form of participation on wikipedia*. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. CSCW '10. pages 127–130. New York, NY, USA. 2010. ACM.
- [Ad09] Adami, M.: *Performance engineering and analysis for a web-based collaboration platform*. Diploma thesis. Fakultät für Informatik, Technische Universität München. 2009.
- [ADR06] Auer, S.; Dietzold, S.; Riechert, T.: *OntoWiki - A Tool for Social, Semantic Collaboration*. In (Cruz, I. F.; Decker, S.; Allemang, D.; Preist, C.; Schwabe, D.; Mika, P.; Uschold, M. et al., Ed.): *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*. volume 4273 of *Lecture Notes in Computer Science*. pages 736–749. Berlin / Heidelberg. 2006. Springer.
- [AL07] Auer, S.; Lehmann, J.: *What Have Innsbruck and Leipzig in Common? Extracting Semantics from Wiki Content* *The Semantic Web: Research and Applications*. In (Franconi, E.; Kifer, M.; May, W., Ed.): *The Semantic Web: Research and Applications*. volume 4519 of *Lecture Notes in Computer Science*. chapter 36, pages 503–517. Springer. Berlin, Heidelberg. 2007.
- [ALZ03] Ancona, D.; Lagorio, G.; Zucca, E.: *Jam - designing a Java extension with mixins*. *ACM Trans. Program. Lang. Syst.* 25(5):641–712. 2003.
- [Am10] Amram, N.; Antonelli, S.; Haywood, S.; Lloyd, S.; Luehring, F.; Poulard, G.: *The use of the TWiki Web in ATLAS*. *Journal of Physics: Conference Series*. 219(8). 2010.

- [AMY08] Ayers, P.; Matthews, C.; Yates, B.: *How Wikipedia Works: And How You Can Be a Part of It*. No Starch Press. September 2008. 159327176X.
- [Ar09] Arnold, C.; Fleming, T.; Largent, D.; Lüer, C.: *DynaTable: a Wiki extension for structured data*. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*. WikiSym '09. pages 26:1–26:2. New York, NY, USA. 2009. ACM.
- [Au05a] Auer, S.: *Powl - A Web Based Platform for Collaborative Semantic Web Development*. In *Proc. of 1st Workshop Scripting for the Semantic Web (SFSW'05), Hersonissos, Greece, May 30, 2005*. May 2005.
- [Au05b] Aumueller, D.: *SHAWN: Structure helps a wiki navigate*. In *Proceedings of the BTW-Workshop WebDB Meets IR*. 2005.
- [Au05c] Aumueller, D.: *Semantic authoring and retrieval within a Wiki*. In *ESWC*. 2005.
- [Au06] Auer, S.: *Powl - Framwork für Entwicklung von semantischen Web-Applikationen / Ontowiki - semantische Community-Kollaborationsplattform*. In (Blumauer, A.; Pellegrini, T., Ed.): *Semantic Web Fibel 06 - Leitfaden und Einstiegsunkte für die Praxis*. Semantic Web School - Zentrum für Wissenstransfer. Wien. 2006.
- [Au08] Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; Ives, Z.: *DBpedia: A Nucleus for a Web of Open Data*. In *Proceedings of the 6th International Semantic Web Conference (ISWC)*. volume 4825 of *Lecture Notes in Computer Science*. pages 722–735. Springer. 2008.
- [Au10] Aumueller, D.: *Usability Meets Instant Gratification on the Semantic Web*. *CoRR*. abs/1011.2386. 2010.
- [Ba09] Barrett, D. J.: *MediaWiki - Wikipedia and beyond*. O'Reilly Media. 2009. 978-0-596-51979-7.
- [BCT07] Bollacker, K. D.; Cook, R. P.; Tufts, P.: *Freebase: A Shared Database of Structured General Human Knowledge*. In *AAAI*. pages 1962–1963. AAAI Press. 2007.
- [BD09] Bruegge, B.; Dutoit, A. H.: *Object Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall. 2009. 0136061257.
- [BG04] Brickley, D.; Guha, R.: *RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation 10 February 2004*. <http://www.w3.org/TR/rdf-schema> (cited 2012-02-04). 2004.
- [BG06] Buffa, M.; Gandon, F.: *SweetWiki: Semantic Web Enabled Technologies in Wiki*. *Proceedings of the international symposium on Symposium on Wikis*, ACM Press. page 10. 2006.
- [BHBL09] Bizer, C.; Heath, T.; Berners-Lee, T.: *Linked Data – The Story So Far*. *International Journal on Semantic Web and Information Systems*. 5(3):1–22. 2009.
- [Bi09] Bizer, C.; Auer, S.; Kobilarov, G.; Lehmann, J.; Becker, C.; Hellmann, S.: *DBpedia - Querying Wikipedia like a Database and An Interlinking-Hub in the Web of Data*. April 2009. Querying Wikipedia Like a Database (4/4/2009) FU Berlin, Universität Leipzig.

-
- [Bi12] Birkmeier, D.; Buckl, S.; Gehlert, A.; Matthes, F.; Neubert, C.; Overhage, S.; Roth, S. et al.: *The Role of Services in Governmental Enterprise Architectures – The Case of the German Federal Government*. Anthopoulos, L.: An Investigative Assessment of the role of Enterprise Architecture in realizing E-Government Transformation. In Saha, P. (Ed.). *Enterprise Architecture for Connected E-Government: Practices and Innovations*. Hershey, PA: IGI Global. 2012. 9781466618244.
- [BL98] Berners-Lee, T.: *Semantic Web Road Map*. 1998.
- [BLC95] Berners-Lee, T.; Connolly, D.: *Hypertext Markup Language - 2.0*. RFC Editor. United States. 1995.
- [BM06] Büchner, T.; Matthes, F.: *Introspective Model-Driven Development*. In *EWSA*. pages 33–49. 2006.
- [BMN09] Büchner, T.; Matthes, F.; Neubert, C.: *A Concept and Service based Analysis of Commercial and Open Source Enterprise 2.0 Tools*. In *International Conference on Knowledge Management and Information Sharing*. pages 37–45. Madeira, Portugal. 2009.
- [BMN10a] Büchner, T.; Matthes, F.; Neubert, C.: *Data model driven implementation of web cooperation systems with Tricia*. In *Proceedings of the Third international conference on Objects and databases*. ICOODB'10. pages 70–84. Berlin, Heidelberg. 2010. Springer.
- [BMN10b] Büchner, T.; Matthes, F.; Neubert, C.: *Functional Analysis of Enterprise 2.0 Tools - a Services Catalog*. In *Lecture Notes of Communications in Computer and Information Science (CCIS)*. 2010.
- [BMS10] Buckl, S.; Matthes, F.; Schweda, C.: *A Generative Approach for Creating Stakeholder-specific Enterprise Architecture Views*. In *Electronic Proceedings of Forum of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE 2010)*. volume 72 of *Lecture Notes in Business Information Processing (LNBIP)*. pages 136–149. Hammamet. 2010. Springer.
- [Bo91] Boehm, B. W.: *Software Risk Management: Principles and Practices*. *IEEE Software*. 8(1):32–41. 1991.
- [Bo04] Boiko, B.: *Content Management Bible (Bible)*. John Wiley & Sons. November 2004. 0764573713.
- [Bo07] Bollacker, K.; Tufts, P.; Pierce, T.; Cook, R.: *A Platform for Scalable, Collaborative, Structured Information Integration*. In *Sixth International Workshop on Information Integration on the Web Association for the Advancement of Artificial Intelligence*. 2007.
- [Bo08] Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; Taylor, J.: *Freebase: a collaboratively created graph database for structuring human knowledge*. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. SIGMOD '08. pages 1247–1250. New York, NY, USA. 2008. ACM.
- [Bo09] Boulos, M.: *Semantic Wikis: A Comprehensible Introduction with Examples from*

- the Health Sciences. Journal of Emerging Technologies in Web Intelligence.* 1(1). 2009.
- [BP08] Baumeister, J.; Puppe, F.: *Web-based Knowledge Engineering using Knowledge Wikis.* In *Proceedings of Symbiotic Relationships between Semantic Web and Knowledge Engineering (AAAI 2008 Spring Symposium)*. 2008.
- [BRP07a] Baumeister, J.; Reutelshoef, J.; Puppe, F.: *KnowWE - Community-based Knowledge Capture with Knowledge Wikis.* In *K-CAP '07: Proceedings of the 4th international conference on Knowledge capture*. pages 189–190. New York, NY, USA. 2007. ACM.
- [BRP07b] Baumeister, J.; Reutelshoef, J.; Puppe, F.: *Markups for Knowledge Wikis.* In (Handschuh, S.; Collier, N.; Groza, T.; Dieng, R.; Sintek, M.; de Waard, A., Ed.): *SAAKM*. volume 289 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2007.
- [BRP11] Baumeister, J.; Reutelshoef, J.; Puppe, F.: *KnowWE: a Semantic Wiki for knowledge engineering.* *Appl. Intell.* 35(3):323–344. 2011.
- [BSZ07] Braun, S.; Schmidt, A.; Zacharias, V.: *SOBOLEO: vom kollaborativen Tagging zur leichtgewichtigen Ontologie.* In (Gross, T., Ed.): *Mensch & Computer - 7. Fachübergreifende Konferenz - M&C 2007*. pages 209–218. München. 2007. Oldenbourg Verlag.
- [Bu96] Buschmann, G.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M.: *Pattern-Oriented Software Architecture: a system of patterns*. volume 1. John Wiley & Sons. 1996.
- [Bü07] Büchner, T.: *Introspektive Modellgetriebene Softwareentwicklung.* PhD thesis. Technische Universität München. 2007.
- [Bu09] Buckl, S.; Matthes, F.; Neubert, C.; Schweda, C. M.: *A Wiki-based Approach to Enterprise Architecture Documentation and Analysis.* In *The 17th European Conference on Information Systems (ECIS) – Information Systems in a Globalizing World: Challenges, Ethics and Practices, 8. – 10. June 2009, Verona, Italy*. pages 2192–2204. Verona, Italy. 2009.
- [Bu10] Buckl, S.; Matthes, F.; Neubert, C.; Schweda, C. M.: *A Lightweight Approach to Enterprise Architecture Modeling and Documentation.* In (Soffer, P.; Proper, E., Ed.): *CAiSE Forum*. volume 72 of *Lecture Notes in Business Information Processing (LNBIP)*. pages 136–149. Springer. 2010.
- [Bu11a] Buffa, M.; Gandon, F.; Sander, P.; Faron, C.; Ereteo, G.: *SweetWiki: a semantic wiki.* *Web Semantics: Science, Services and Agents on the World Wide Web.* 6(1). 2011.
- [Bu11b] Buneman, P.; Cheney, J.; Lindley, S.; Müller, H.: *The database Wiki project: a general-purpose platform for data curation and collaboration.* *SIGMOD Record.* 40(3):15–20. 2011.
- [Bu11c] Buneman, P.; Cheney, J.; Lindley, S.; Müller, H.: *DBWiki: a structured wiki for curated data and collaborative data management.* In (Sellis, T. K.; Miller, R. J.; Kementsietsidis, A.; Velegrakis, Y., Ed.): *SIGMOD Conference*. pages 1335–1338.

- ACM. 2011.
- [Ca11] Casagni, C.; Francescomarino, C. D.; Dragoni, M.; Fiorentini, L.; Franci, L.; Gerosa, M.; Ghidini, C. et al.: *Wiki-Based Conceptual Modeling: An Experience with the Public Administration*. In (Aroyo, L.; Welty, C.; Alani, H.; Taylor, J.; Bernstein, A.; Kagal, L.; Noy, N. F. et al., Ed.): *International Semantic Web Conference (2)*. volume 7032 of *Lecture Notes in Computer Science*. pages 17–32. Springer. 2011.
- [CCT04] Campanini, S. E.; Castagna, P.; Tazzoli, R.: *Platypus Wiki: a Semantic Wiki Wiki Web*. In *Semantic Web Applications and Perspectives, Proceedings of 1st Italian SemanticWeb Workshop*. December 2004.
- [Ch08a] Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W. C.; Wallach, D. A.; Burrows, M.; Chandra, T. et al.: *Bigtable: A Distributed Storage System for Structured Data*. *ACM Trans. Comput. Syst.* 26:1–26. June 2008.
- [Ch08b] Chu, S. K.-W.: *TWiki for knowledge building and management*. *Online Information Review*. 32(6):745–758. 2008.
- [Da08] Damiani, E.; Ceravolo, P.; Corallo, A.; Elia, G.; Zilli, A.: *KIWI: A Framework for Enabling Semantic Knowledge Management*. In *Semantic Knowledge Management: An Ontology-Based Framework*. pages 1–23. Idea Group Reference. 2008.
- [Da11] Dacka, P.: *Functional Analysis of Enterprise 2.0 Tools - Expectations, Experiences and Valuations from an End-User Perspective*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2011.
- [DAR06] Dietzold, S.; Auer, S.; Riechert, T.: *Kollaborative Wissensarbeit mit OntoWiki*. In *Proceedings of the INFORMATIK 2006 Workshop: Bildung von Sozialen Netzwerken in Anwendungen der "Social Software"*. 2006.
- [DC10] Dirolf, M.; Chodorow, K.: *MongoDB: The Definitive Guide*. O'Reilly Media. 1 edition. 2010. 1449381561.
- [De05a] Decker, B.; Ras, E.; Rech, J.; Klein, B.; Hoecht, C.: *Self-organized Reuse of Software Engineering Knowledge Supported by Semantic Wikis*. *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE), held at the 4th International Semantic Web Conference (ISWC 2005) November 6th-10th*. 2005.
- [De05b] Decker, B.; Ras, E.; Rech, J.; Klein, B.; Reuschling, C.; Höcht, C.; Kilian, L. et al.: *A Framework for Agile Reuse in Software Engineering using Wiki Technology*. In (Althoff, K.-D.; Dengel, A.; Bergmann, R.; Nick, M.; Roth-Berghofer, T., Ed.): *Wissensmanagement*. pages 411–414. DFKI. 2005.
- [DH10] Dengler, F.; Happel, H.-J.: *Collaborative modeling with semantic MediaWiki*. In (Ayers, P.; Ortega, F., Ed.): *Int. Sym. Wikis*. ACM. 2010.
- [DIVZ08] Di Iorio, A.; Vitali, F.; Zacchiroli, S.: *Wiki content templating*. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*. pages 615–624. New York, NY, USA. 2008. ACM.

- [DIZ06] Di Iorio, A.; Zacchiroli, S.: *Constrained Wiki: an Oxymoron?* In *Proceedings of the 2006 international symposium on Wikis*. WikiSym '06. pages 89–98. New York, NY, USA. 2006. ACM.
- [Dr07] Drakos, N.: *Magic Quadrant for Team Collaboration and Social Software*. Gartner Research, ID Number: G00151493. 2007.
- [DST06] Dello, K.; Simperl, E. P. B.; Tolksdorf, R.: *Creating and using Semantic Web information with Makna*. In (Völkel, M.; Schaffert, S., Ed.): *SemWiki*. volume 206 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2006.
- [DSW06] Davies, J.; Studer, R.; Warren, P., Ed. *Semantic Web Technologies: Trends and Research in Ontology-Based Systems*. John Wiley & Sons. Chichester, UK. 2006. 978-0-470-02596-3.
- [Du05] Dueck, G.; Ebersbach, A.; Glaser, M.; Heigl, R.; Adelung, A.: *Wiki: Web Collaboration*. Springer. Secaucus, NJ, USA. 2005. 3540259953.
- [EM00] Edmunds, A.; Morris, A.: *The problem of information overload in business organisations: a review of the literature*. *International Journal of Information Management*. 20(1):17–28. 2000.
- [ET11] Embley, D. W.; Thalheim, B.: *Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges*. Springer. 2011. 3642158641.
- [Fe11] Fensel, D.; Facca, F. M.; Simperl, E.; Toma, I.; Fensel, D.; Facca, F. M.; Simperl, E. et al.: *Semantic Web*. In *Semantic Web Services*. pages 87–104. Springer. 2011. 10.1007/978-3-642-19193-0_6.
- [Fo03] Fowler, M.: *Patterns of Enterprise Application Architecture*. Addison-Wesley. 2003. With contributions from Rice, D., Foemmel, M., Hieatt, E., Mee, R. and Stafford, R.
- [Fr98] Froehlich, G.; Hoover, H.; Liu, L.; Sorenson, P.: *Designing object-oriented frameworks*. *CRC Handbook of Object Technology*. pages 25:1–25:21. 1998.
- [Ga10a] García, R.; Gil, R.; Gimeno, J. M.; Granollers, T.; López, J. M.; Oliva, M.; Pascual, A.: *Semantic wiki for quality management in software development projects*. *IET Software*. 4(6):1–19. 2010.
- [Ga10b] Gassler, W.; Zangerle, E.; Tschuggnall, M.; Specht, G.: *SnoopyDB: narrowing the gap between structured and unstructured information using recommendations*. In *Proceedings of the 21st ACM conference on Hypertext and hypermedia*. HT '10. pages 271–272. New York, NY, USA. 2010. ACM.
- [GH05] Golder, S. A.; Huberman, B. A.: *The Structure of Collaborative Tagging Systems*. *CoRR*. abs/cs/0508082. 2005.
- [Gh07] Ghali, A. E.; Tifous, A.; Buffa, M.; Giboin, A.; Dieng-Kuntz, R.: *Using a Semantic Wiki in Communities of Practice*. In *2nd Intern. Workshop on Building Technology Enhanced Learning Solutions for Communities of Practice*. 2007.
- [Gh08] Ghidini, C.; Rospocher, M.; Serafini, L.; Kump, B.; Pammer, V.; Faatz, A.; Zinnen, A. et al.: *Collaborative Knowledge Engineering via Semantic MediaWiki*. In

- (Auer, S.; Schaffert, S.; Pellegrini, T., Ed.): *International Conference on Semantic Systems (I-SEMANTICS '08)*. pages 134–141. Graz, Austria. September 2008.
- [Gh09] Ghidini, C.; Kump, B.; Lindstaedt, S.; Mahbub, N.; Pammer, V.; Rospocher, M.; Serafini, L.: *MoKi: The Enterprise Modelling Wiki*. In (Aroyo, L.; Traverso, P.; Ciravegna, F.; Cimiano, P.; Heath, T.; Hyvönen, E.; Mizoguchi, R. et al., Ed.): *The Semantic Web: Research and Applications*. volume 5554. chapter 65, pages 831–835. Springer. Berlin, Heidelberg. 2009.
- [GL10] Gläser, J.; Laudel, G.: *Experteninterviews und qualitative Inhaltsanalyse: als Instrumente rekonstruierender Untersuchungen*. VS Verlag für Sozialwissenschaften. 4th edition. 2010. 978-3531172385.
- [GMM03] Guha, R.; McCool, R.; Miller, E.: *Semantic search*. pages 700–709. 2003.
- [GP01] Genero, M.; Piattini, M.: *Empirical validation of measures for class diagram structural complexity through controlled experiments*. *5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*. 2001.
- [Gr12] Großmann, A.: *Usage of HybridWikis for Business Processes Support - A Case Study in a Medium-Sized Enterprise*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2012.
- [GZS11] Gassler, W.; Zangerle, E.; Specht, G.: *The Snoopy Concept: Fighting Heterogeneity in Semistructured and Collaborative Information Systems by using Recommendations*. In *International Conference on Collaboration Technologies and Systems*. Philadelphia, PE. 2011.
- [Ha04] Hayes, P.: *RDF Semantics W3C Recommendation 10 February 2004*. <http://www.w3.org/TR/rdf-mt> (cited 2012-02-04). 2004.
- [Ha09] Harrington, J.: *Relational Database Design and Implementation: Clearly Explained*. Elsevier. 3 edition. 2009. 0123747309.
- [Ha10] Hahn, R.; Bizer, C.; Sahnwaldt, C.; Herta, C.; Robinson, S.; Bürgle, M.; Düwiger, H. et al.: *Faceted Wikipedia Search*. In (Abramowicz, W.; Tolksdorf, R., Ed.): *BIS*. volume 47 of *Lecture Notes in Business Information Processing*. pages 1–11. Springer. 2010.
- [HBS06] Hepp, M.; Bachlechner, D.; Siorpaes, K.: *OntoWiki: community-driven ontology engineering and ontology usage based on Wikis*. In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*. pages 143–144. New York, NY, USA. 2006. ACM.
- [He04] Hevner, A. R.; March, S. T.; Park, J.; Ram, S.: *Design Science in Information Systems Research*. *MIS Quarterly*. 28(1):75–105. 2004.
- [HGT11] Hamasaki, M.; Goto, M.; Takeda, H.: *Social Infobox: collaborative knowledge construction by social property tagging*. In (Hinds, P. J.; Tang, J. C.; Wang, J.; Bardram, J. E.; Ducheneaut, N., Ed.): *CSCW*. pages 641–644. ACM. 2011.
- [HL09] Hoenderboom, B.; Liang, P.: *A Survey of Semantic Wikis for Requirements Engineering Architecture*. 2009.

- [HLS05] Haake, A.; Lukosch, S.; Schümmer, T.: *Wiki-templates: adding structure support to wikis on demand*. In *WikiSym '05: Proceedings of the 2005 international symposium on Wikis*. pages 41–51. New York, NY, USA. 2005. ACM.
- [Ho09] Hoehndorf, R.; Bacher, J.; Backhaus, M.; Gregorio, S.; Loebe, F.; Pruffer, K.; Uciteli, A. et al.: *BOWiki: an ontology-based wiki for annotation of data and integration of knowledge in biology*. *BMC Bioinformatics*. 10(Suppl 5). 2009.
- [HSP09] Hansch, D.; Schnurr, H.-P.; Pissierssens, P.: *Semantic MediaWiki+ als Wissensplattform für Unternehmen*. In (Hinkelmann, K.; Wache, H., Ed.): *Wissensmanagement*. volume 145 of *LNI*. pages 211–221. GI. 2009.
- [Hu11] Hunecker, M.: *A Comparison of Web-based Approaches for Structuring Information - A Literature Analysis*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2011.
- [HV09] Hagemann, S.; Vossen, G.: *Categorizing User-Generated Content (extended abstract)*. *Proceedings of the WebSci'09: Society On-Line*. 2009.
- [HZG09] Hoffart, J.; Zesch, T.; Gurevych, I.: *An architecture to support intelligent user interfaces for Wikis by means of Natural Language Processing*. In (Riehle, D.; Bruckman, A., Ed.): *Int. Sym. Wikis*. ACM. 2009.
- [IB07] Isbell, J.; Butler, M. H.: *Extracting and Re-using Structured Data from Wikis*. Technical Report HPL-2007-182. Hewlett-Packard. 2007.
- [JH11] Jones, P. L.; Høimyr, N.: *TWiki a collaboration tool for the LHC*. In (Ortega, F.; Forte, A., Ed.): *Int. Sym. Wikis*. pages 207–208. ACM. 2011.
- [JZW09] Jing, Y.; Zhang, C.; Wang, X.: *An Empirical Study on Performance Comparison of Lucene and Relational Database*. In *Proceedings of the 2009 International Conference on Communication Software and Networks*. ICCSN '09. pages 336–340. Washington, DC, USA. 2009. IEEE Computer Society.
- [KC04] Klyne, G.; Carroll, J. J.: *Resource Description Framework (RDF): Concepts and Abstract Syntax*. <http://www.w3.org/TR/rdf-concepts> (cited 2012-02-04). 2004.
- [KGI08] Kramer, M.; Gregorowicz, A.; Iyer, B.: *Wiki trust metrics based on phrasal analysis*. In *Proceedings of the 4th International Symposium on Wikis*. WikiSym '08. pages 24:1–24:10. New York, NY, USA. 2008. ACM.
- [KHD05] Klein, B.; Höcht, C.; Decker, B.: *Beyond Capturing and Maintaining Software Engineering Knowledge-“Wikitology” as Shared Semantics*. In *Workshop on Knowledge Engineering and Software Engineering, KI*. 2005.
- [Ki06] Kiesel, M.: *Kaukolu: Hub of the Semantic Corporate Intranet*. In (Völkel, M.; Schaffert, S., Ed.): *SemWiki*. volume 206 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2006.
- [Ki08] Kiesel, M.; Schwarz, S.; van Elst, L.; Buscher, G.: *Mymory: Enhancing a Semantic Wiki with Context Annotations*. In (Bechofer, S.; Hauswirth, M.; Hoffmann, J.; Koubarakis, M., Ed.): *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain*,

- June 1-5, 2008. volume 5021 of *LNAI*. pages 817–821. Springer. 2008.
- [Ko01] Koch, C.: *Data Integration against Multiple Evolving Autonomous Schemata*. PhD thesis. Technische Universität Wien. 2001.
- [Ko09] Kobilarov, G.; Bizer, C.; Auer, S.; Lehmann, J.: *DBpedia - A Linked Data Hub and Data Source for Web Applications and Enterprises*. In *Proceedings of Developers Track of 18th International World Wide Web Conference (WWW 2009), April 20th-24th, Madrid, Spain*. April 2009.
- [Kr95] Krogstie, J.: *Conceptual Modeling for Computerized Information Systems Support in Organizations*. PhD thesis. University of Trondheim. 1995.
- [KS08] Kuhn, T.; Schwitter, R.: *Writing Support for Controlled Natural Languages*. In *Proceedings ALTA 2008*. pages 46–54. 2008.
- [KSV07] Krötzsch, M.; Schaffert, S.; Vrandečić, D.: *Reasoning in Semantic Wikis*. In (Antonioni, G.; Aßmann, U.; Baroglio, C.; Decker, S.; Henze, N.; Patranjan, P.-L.; Tolksdorf, R., Ed.): *Reasoning Web*. volume 4636 of *Lecture Notes in Computer Science*. pages 310–329. Springer. 2007.
- [KT04] Klein, B.; Traphöner, R.: *A Practical Application of Ontologies for Knowledge Sharing and Trading*. In (Abecker, A.; Bickel, S.; Brefeld, U.; Drost, I.; Henze, N.; Herden, O.; Minor, M. et al., Ed.): *LWA*. pages 259–266. Humboldt-Universität Berlin. 2004.
- [Ku08a] Kuhn, T.: *AceWiki: A Natural and Expressive Semantic Wiki*. *CoRR*. abs/0807.4618. 2008.
- [Ku08b] Kuhn, T.: *AceWiki: Collaborative Ontology Management in Controlled Natural Language*. *CoRR*. abs/0807.4623. 2008.
- [Ku08c] Kuhn, T.: *Combining Semantic Wikis and Controlled Natural Language*. In (Bizer, C.; Joshi, A., Ed.): *International Semantic Web Conference (Posters & Demos)*. volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2008.
- [Ku09] Kuhn, T.: *How Controlled English can Improve Semantic Wikis*. In (0002, C. L.; Schaffert, S.; Skaf-Molli, H.; Völkel, M., Ed.): *SemWiki*. volume 464 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2009.
- [KVV05] Krötzsch, M.; Vrandečić, D.; Völkel, M.: *Wikipedia and the Semantic Web-The Missing Links*. *Proceedings of Wikimania*. 2005. 2005.
- [KVV06] Krötzsch, M.; Vrandečić, D.; Völkel, M.: *Semantic MediaWiki*. In (Cruz, I.; Decker, S.; Allemang, D.; Preist, C.; Schwabe, D.; Mika, P.; Uschold, M. et al., Ed.): *The Semantic Web - ISWC 2006*. volume 4273 of *Lecture Notes in Computer Science*. pages 935–942. Springer. 2006.
- [La07] Lange, C.: *Towards Scientific Collaboration in a Semantic Wiki*. In *Bridging the Gap between Semantic Web and Web 2.0 (SemNet 2007)*. pages 119–126. 2007.
- [La09] Lankhorst, M., Ed. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer. Berlin. 2. edition. 2009. 978-3-642-01309-6.

- [La10] Lange, C.: *SWiM – A Semantic Wiki for Mathematical Knowledge Management*. *CoRR*. abs/1003.5196. 2010.
- [La11] Lange, C.: *Enabling Collaboration on Semiformal Mathematical Knowledge by Semantic Web Integration*. PhD thesis. Jacobs University Bremen. 2011.
- [LC01] Leuf, B.; Cunningham, W.: *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley. April 2001. 020171499X.
- [Le09] Lehmann, J.; Bizer, C.; Kobilarov, G.; Auer, S.; Becker, C.; Cyganiak, R.; Hellmann, S.: *DBpedia - A Crystallization Point for the Web of Data*. *Journal of Web Semantics*. 7(3):154–165. 2009.
- [LFL05] Lio, E. D.; Fraboni, L.; Leo, T.: *TWiki-based facilitation in a newly formed academic community of practice*. In (Riehle, D., Ed.): *Int. Sym. Wikis*. pages 85–111. ACM. 2005.
- [LFZ09] Li, G.; Feng, J.; Zhou, L.: *Interactive search in XML data*. In *Proceedings of the 18th international conference on World wide web*. WWW '09. pages 1063–1064. New York, NY, USA. 2009. ACM.
- [Li09] Lincoln, S. R.: *Mastering Web 2.0: Transform Your Business Using Key Website and Social Media Tools*. Kogan Page Ltd. London, UK, UK. 2009. 0749454660, 9780749454661.
- [LK06] Lange, C.; Kohlhase, M.: *A Semantic Wiki for Mathematical Knowledge Management*. In (Völkel, M.; Schaffert, S., Ed.): *Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics*. Workshop on Semantic Wikis. ESWC2006. June 2006.
- [LKL10] Lucke, C.; Krell, S.; Lechner, U.: *Critical Issues in Enterprise Architecting - A Literature Review*. In (Santana, M.; Luftman, J. N.; Vinze, A. S., Ed.): *AMCIS*. page 305. Association for Information Systems. 2010.
- [LM10] Lakshman, A.; Malik, P.: *Cassandra: a decentralized structured storage system*. *SIGOPS Oper. Syst. Rev.* 44:35–40. April 2010.
- [LOZ04] Ludwig, L.; O’Sullivan, D.; Zhou, X.: *Artificial Memory Prototype for Personal Semantic Subdocument Knowledge Management*. In *3rd International Semantic Web Conference (ISWC2004)*. Hiroshima, Japan. 2004.
- [LSA11] Lubbers, P.; Salim, F.; Albers, B.: *Pro Html5 Programming*. Springer. 2011. 143023864X.
- [Lu05] Ludwig, L.: *Semantic personal knowledge management*. Technical report. DERI Galway. 2005.
- [Lu09] Ludwig, L.: *Artificial Memory - Eine kurze Einführung in Struktur, Aufgaben und Erfolgskennzahlen*. In (Knut Hinkelmann, H. W., Ed.): *Lecture Notes in Informatics WM2009: 5th Conference on Professional Knowledge Management, Solothurn (2009)*. 2009.
- [Ma03] Martin, R. C.: *Agile software development: principles, patterns, and practices*. Prentice Hall. 2003.

-
- [Ma05] Malik, S.: *Enterprise Dashboards: Design and Best Practices for IT*. John Wiley & Sons. 2005. 0471738069.
- [Ma08] Matthes, F.; Buckl, S.; Leitel, J.; Schweda, C. M.: *Enterprise Architecture Management Tool Survey 2008*. Chair for Informatics 19 (sebis), Technische Universität München. Munich, Germany. 2008.
- [Ma11] Maier, M.: *Interactive Management of Attributes with Types, Formats, and Constraints in an Enterprise 2.0 Tool - Design and Implementation*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2011.
- [Mc06] McAfee, A. P.: *Enterprise 2.0: The Dawn of Emergent Collaboration*. *MIT Sloan Management Review*. 47(3):21–28. 2006.
- [Mc09] McAfee, A.: *Enterprise 2.0: New Collaborative Tools for Your Organization's Toughest Challenges*. Harvard Business Press. 1 edition. 2009. 1422125874.
- [MCS09] Matthes, F.; C., N.; Steinhoff, A.: *Federated Application Lifecycle Management Based on an Open Web Architecture*. In *Workshop Design for Future - Langlebige Softwaresysteme, GI-Arbeitskreis Langlebige Softwaresysteme (L2S2)*. Karlsruhe, Germany. 2009.
- [MH04] McGuinness, D. L.; van Harmelen, F.: *OWL Web Ontology Language Overview W3C Recommendation 10 February 2004*. <http://www.w3.org/TR/owl-features> (cited 2012-02-04). 2004.
- [MHG10] McCandless, M.; Hatcher, E.; Gospodnetic, O.: *Lucene in Action*. Manning Publications. 2 edition. May 2010. 1933988177.
- [Mi08] Millard, D. E.; Bailey, C.; Boulain, P.; Chennupati, S.; Davis, H. C.; Howard, Y. M.; Wills, G.: *Semantics on demand: Can a Semantic Wiki replace a knowledge base? The New Review of Hypermedia and Multimedia*. 14(1):95–120. 2008.
- [Mi10] Mirbeth, A.: *Review and extension of the sebis Enterprise 2.0 Tool survey*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2010.
- [MN11a] Matthes, F.; Neubert, C.: *Enabling Knowledge Workers to Collaboratively Add Structure to Enterprise Wikis*. In *Proceedings of the 12th European Conference on Knowledge Management*. pages 617–625. 2011.
- [MN11b] Matthes, F.; Neubert, C.: *Wiki4EAM: Using Hybrid Wikis for Enterprise Architecture Management*. In *7th International Symposium on Wikis and Open Collaboration (WikiSym)*. Mountain View, California, USA. 2011.
- [MN12] Matthes, F.; Neubert, C.: *Hybride Wikis als Repository für IT-Unternehmensarchitektur*. chapter 5.4.2, pages 174–182. Dpunkt. 2012.
- [MNS11] Matthes, F.; Neubert, C.; Steinhoff, A.: *Hybrid Wikis: Empowering Users to Collaboratively Structure Information*. In *Proceedings of the 6th International Conference on Software and Data Technologies*. pages 250–259. 2011.
- [MNS12] Matthes, F.; Neubert, C.; Steinhoff, A.: *Structuring Folksonomies with Implicit Tag Relations*. In *Proceedings of the 23rd ACM conference on Hypertext and hypermedia*. 2012.

- [NS06] Nixon, L. J. B.; Simperl, E. P. B.: *Makna and MultiMakna: towards semantic and multimedia capability in wikis for the emerging web*. In *Semantics'06*. Vienna, Austria. 2006.
- [NST09a] Neubert, C.; Stecher, M.; Taing, S.: *Enterprise 2.0 - Eine Typologisierung*. In *Mensch und Computer 2009 - Workshop-Tagungsband, Enterprise 2.0 - Web 2.0 im Unternehmen*. Berlin. 2009.
- [NST09b] Neubert, C.; Stecher, M.; Taing, S.: *Typology Approaches for Enterprise 2.0 Applications and Technologies*. In *IWM / ZEW workshop, The potential of social software for knowledge creation and economic performance*. Mannheim. 2009.
- [O'05] O'Reilly, T.: *What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software*. <http://www.oreillyn.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>. September 2005. Stand 12.5.2009.
- [OBD06] Oren, E.; Breslin, J. G.; Decker, S.: *How semantics make better wikis*. In (Carr, L.; Roure, D. D.; Iyengar, A.; Goble, C. A.; Dahlin, M., Ed.): *WWW*. pages 1071–1072. ACM. 2006.
- [Or05] Oren, E.: *SemperWiki: a semantic personal Wiki*. In *Proc. of 1st WS on The Semantic Desktop, Galway, Ireland*. 2005.
- [Or06a] Oren, E.; Delbru, R.; Möller, K.; Völkel, M.; Handschuh, S.: *Annotation and navigation in semantic wikis*. In *Proceedings of the SemWiki2006 - From Wiki to Semantics Workshop, in conjunction with ESWC2006. Budva, Montenegro*. June 2006.
- [Or06b] Oren, E.; Völkel, M.; Breslin, J. G.; Decker, S.: *Semantic Wikis for Personal Knowledge Management. Lecture Notes in Computer Science*. 4080:509–518. 2006.
- [Pa09] Paschke, A.; Coskun, G.; Luczak-Rösch, M.; Oldakowski, R.; Harasic, M.; Heese, R.; Schäfermeier, R. et al.: *Realizing the Corporate Semantic Web: Concept Paper*. Technical report. Freie Universität Berlin. Berlin. April 2009.
- [Pa11] Passant, A.: *Semantic Web Technologies for Enterprise 2.0*. IOS Press. 2011.
- [Pe11] Peter; Cheney, J.; Lindley, S.; Müller, H.: *Using Links to prototype a Database Wiki. 13th International Symposium on Database Programming Languages*. 2011.
- [Pf08] Pfisterer, F.; Nitsche, M.; Jameson, A.; Barbu, C.: *User-Centered Design and Evaluation of Interface Enhancements to the Semantic MediaWiki*. In *Workshop on Semantic Web User Interaction at CHI 2008*. 2008.
- [PM07] Price, J.; McClain, L.: *Oracle Database 11g SQL: Master SQL and PL/SQL in the Oracle Database*. McGraw-Hill Professional. 2007. 0071498508.
- [Po07] Powers, S.: *Adding Ajax. Making Existing Sites More Interactive*. O'Reilly Media. 2007. 0596529368.
- [PS08] Prud'hommeaux, E.; Seaborne, A.: *SPARQL Query Language for RDF W3C Recommendation 15 January 2008*. <http://www.w3.org/TR/rdf-sparql-query> (cited 2012-02-04). 2008.

-
- [PT11] Priedhorsky, R.; Terveen, L. G.: *Wiki grows up: arbitrary data models, access control, and beyond*. In (Ortega, F.; Forte, A., Ed.): *Int. Sym. Wikis*. pages 63–71. ACM. 2011.
- [Ra08] Rauschmayer, A.: *Next-Generation Wikis: What Users Expect; How RDF Helps*. In (0002, C. L.; Schaffert, S.; Skaf-Molli, H.; Völkel, M., Ed.): *SemWiki*. volume 360 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2008.
- [Ra10a] Rauschmayer, A.: *Connected Information Management*. PhD thesis. Ludwig-Maximilians-Universität München. 2010.
- [Ra10b] Rauschmayer, A.: *Structure your wiki: Improving support for structured data in wikis*. Technical report. Ludwig-Maximilians-Universität München, Institut für Informatik. München. 2010.
- [Ra11] Rau, K.: *Analysis and Evaluation of the Model Accuracy in Hybrid Wikis*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2011.
- [RBP08] Reutelshoefer, J.; Baumeister, J.; Puppe, F.: *Ad-Hoc Knowledge Engineering with Semantic Knowledge Wikis*. In *SemWiki'08: Proceedings of 3rd Semantic Wiki workshop - The Wiki Way of Semantics (CEUR Proceedings 360)*. 2008.
- [Re10] Reutelshoefer, J.; Lemmerich, F.; Baumeister, J.; Wintjes, J.; Haas, L.: *Taking OWL to Athens – Semantic Web Technology takes Ancient Greek History to Students*. In *ESWC'10: Proceedings of the 7th Extended Semantic Web Conference*. pages 333–347. Springer. 2010.
- [RG02] Raygan, R.; Green, D.: *Internet collaboration: TWiki*. In *SoutheastCon, 2002. Proceedings IEEE*. pages 137–141. 2002.
- [RK06] Rauschmayer, A.; Kammergruber, W. C.: *A Wiki as an Extensible RDF Presentation Engine*. In *ESWC Wsh. Semantic Wikis—From Wiki to Semantics*. June 2006.
- [Ro08] Rospocher, M.; Ghidini, C.; Serafini, L.; Kump, B.; Pammer, V.; Lindstaedt, S.; Faatz, A. et al.: *Collaborative Enterprise Integrated Modelling*. In (Gangemi, A.; Keizer, J.; Presutti, V.; Stoermer, H., Ed.): *Proceedings of the 5th Workshop on Semantic Web Applications and Perspectives (SWAP2008)*. CEUR Workshop Proceedings. Rome, Italy. December 2008. online http://ceur-ws.org/Vol-426/swap2008_submission_62.pdf.
- [Ro09] Rospocher, M.; Ghidini, C.; Pammer, V.; Serafini, L.; Lindstaedt, S. N.: *MoKi: the Modelling wiKi*. In (0002, C. L.; Schaffert, S.; Skaf-Molli, H.; Völkel, M., Ed.): *SemWiki*. volume 464 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2009.
- [RS04] Radziwill, N.; Shelton, A.: *TWiki as a Platform for Collaborative Software Development Management. Proc. SPIE 5496, Glasgow Scotland*. 2004.
- [RW11] Reschenhofer, T.; Walzl, B.: *Enabling Collaborative Information Management on Federated Data Sources Analysis, Design and Prototypical Implementation for MS-SharePoint and MS-Exchange*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2011.

- [Sc06] Schaffert, S.: *IkeWiki: A Semantic Wiki for Collaborative Knowledge Management*. In *WETICE*. pages 388–396. IEEE Computer Society. 2006.
- [Sc08] Schaffert, S.; Bry, F.; Baumeister, J.; Kiesel, M.: *Semantic Wikis*. *IEEE Software*. 25(4):8–11. 2008.
- [Sc09] Schaffert, S.; Eder, J.; Grünwald, S.; Kurz, T.; Sint, R.; Stroka, S.: *KiWi – A Platform for Semantic Social Software*. In *SemWiki2009: Proceedings of the Fourth Workshop on Semantic Wikis*. June 2009.
- [SGW05] Schaffert, S.; Gruber, A.; Westenthaler, R.: *A Semantic Wiki for collaborative knowledge formation*. presentation. 2005.
- [Si09] Sint, R.; Stroka, S.; Schaffert, S.; Ferstl, R.: *Combining Unstructured, Fully Structured and Semi-Structured Information in Semantic Wikis*. In (0002, C. L.; Schaffert, S.; Skaf-Molli, H.; Völkel, M., Ed.): *SemWiki*. volume 464 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2009.
- [SL90] Sheth, A. P.; Larson, J. A.: *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*. *ACM Comput. Surv.* 22(3):183–236. 1990.
- [So05] Souzis, A.: *Building a Semantic Wiki*. *IEEE Intelligent Systems*. 20(5):87–91. 2005.
- [So06] Souzis, A.: *Bringing the "Wiki-Way" to the Semantic Web with Rhizome*. In (Völkel, M.; Schaffert, S., Ed.): *Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics*. Workshop on Semantic Wikis. ESWC2006. June 2006.
- [SS11] Schneider, A.; Steinhoff, A.: *Applying Web Analytics Tools in the Context of Enterprise Social Software*. In *11th European Conference on Knowledge Management (ECKM 2011)*. Passau, Germany. 2011.
- [St08] Steinberg, D.; Budinsky, F.; Paternostro, M.; Merks, E.: *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley. 2 edition. January 2008. 0321331885.
- [SWG06] Schaffert, S.; Westenthaler, R.; Gruber, A.: *IkeWiki: A user-friendly semantic wiki*. In *3rd European Semantic Web Conference (ESWC06)*. 2006.
- [TCC04] Tazzoli, R.; Castagna, P.; Campanini, S. E.: *Towards a Semantic WikiWikiWeb*. In *3rd International Semantic Web Conference (ISWC2004)*. Hiroshima, Japan. 2004.
- [Te09] Textuality, T. B.; Hollander, D.; Layman, A.; Tobin, R.; Thompson, H. S.: *Namespaces in XML 1.0 (Third Edition) W3C Recommendation 8 December 2009*. <http://www.w3.org/TR/REC-xml-names> (cited 2012-02-04). 2009.
- [TFH10] Tramp, S.; Frischmuth, P.; Heino, N.: *OntoWiki – a Semantic Data Wiki Enabling the Collaborative Creation and (Linked Data) Publication of RDF Knowledge Bases*. In (Corcho, O.; Voelker, J., Ed.): *Demo Proceedings of the EKAW 2010*. October 2010.
- [TGP11] Talas, J.; Gregar, T.; Pitner, T.: *Semantic Wiki in Environmental Project Man-*

- agement. In *ISESS*. pages 437–444. 2011.
- [Tr12] Tremmel, M.: *Usage Analysis of Structuring Techniques in Hybrid Wikis based on Application Data*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2012.
- [Ut11] Utz, A.: *Extended UML Class Diagrams for Modeling Emergent Information Structures - Analysis and Prototypical Web-based Implementation*. Bachelor's thesis. Fakultät für Informatik, Technische Universität München. 2011.
- [Vö06] Völkel, M.; Krötzsch, M.; Vrandečić, D.; Haller, H.; Studer, R.: *Semantic Wikipedia*. In *Proceedings of the 15th international conference on World Wide Web*. WWW '06. pages 585–594. New York, NY, USA. 2006. ACM.
- [VWD04] Viégas, F. B.; Wattenberg, M.; Dave, K.: *Studying cooperation and conflict between authors with history flow visualizations*. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. CHI '04. pages 575–582. New York, NY, USA. 2004. ACM.
- [WAM02] Widenius, M.; Axmark, D.; MySQL, A. B.: *MySQL Reference Manual*. O'Reilly Media. 1 edition. 2002. 0596002653.
- [We09] Weikum, G.; Kasneci, G.; Ramanath, M.; Suchanek, F.: *Database and Information-Retrieval Methods for Knowledge Discovery*. *Communications of the ACM*. 52(4):56–64. 2009.
- [WG07] Witte, R.; Gitzinger, T.: *Connecting wikis and natural language processing systems*. In (Désilets, A.; Biddle, R., Ed.): *Int. Sym. Wikis*. pages 165–176. ACM. 2007.
- [WW08] Wu, F.; Weld, D. S.: *Automatically refining the wikipedia infobox ontology*. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*. pages 635–644. New York, NY, USA. 2008. ACM.
- [Yi08] Yitzhak, O. B.; Golbandi, N.; Har'El, N.; Lempel, R.; Neumann, A.; Koifman, S. O.; Sheinwald, D. et al.: *Beyond basic faceted search*. In *Proceedings of the international conference on Web search and web data mining*. WSDM '08. pages 33–44. New York, NY, USA. 2008. ACM.
- [ZB07] Zacharias, V.; Braun, S.: *SOBOLEO – Social Bookmarking and Lightweight Engineering of Ontologies*. In (Noy, N. F.; Alani, H.; Stumme, G.; Mika, P.; Sure, Y.; Vrandečić, D., Ed.): *CKC*. volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2007.
- [ZG10] Zangerle, E.; Gassler, W.: *Recommendation-Based Evolvement of Dynamic Schemata in Semistructured Information Systems*. In (Balke, W.-T.; Lofi, C., Ed.): *Grundlagen von Datenbanken*. volume 581 of *CEUR Workshop Proceedings*. CEUR-WS.org. 2010.
- [ZGS10] Zangerle, E.; Gassler, W.; Specht, G.: *Recommending structure in collaborative semistructured information systems*. In *Proceedings of the fourth ACM conference on Recommender systems*. RecSys '10. pages 261–264. New York, NY, USA. 2010. ACM.

Bibliography

- [ZMG08] Zesch, T.; Müller, C.; Gurevych, I.: *Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary*. In *LREC*. European Language Resources Association. 2008.

Nomenclature

ACE	Attempto Controlled English
ACL	Access Control List
AJAX	Asynchronous JavaScript and XML
API	Application programming interface
BPMN	Business Process Model and Notation
CMDB	Configuration Management Database
CRM	Customer-Relationship-Management
EA	Enterprise Architecture
EAM	Enterprise Architecture Management
EMF	Eclipse Modeling Framework
EWS	Exchange Web Service
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IC	Infrastructure Component
JSON	JavaScript Object Notation JSON
MVC	Model-View-Controller
NLP	Natural Language Processing

Bibliography

OCL	Object Constraint Language
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
REST	Representational state transfer
RSS	Really Simple Syndication
sebis	Software Engineering for Business Information Systems
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SVG	Scalable Vector Graphics
SyCa	System Cartography
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language