FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

# Design and implementation of a task-centric social content management application for end-users

Michael Ostner

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

# Design and implementation of a task-centric social content management application for end-users

# Entwurf und Implementierung einer aufgabenorientierten Social Content Management Anwendung für Endbenutzer

| | |
|---|---|
| Author: | Michael Ostner |
| Supervisor: | Prof. Dr. Florian Matthes |
| Advisor: | M. Sc. Felix Michel |
| Date: | January 15, 2016 |

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, January 14, 2016 Michael Ostner

# Acknowledgments

First of all I would like thank my advisor Felix Michel for his support and patience. Secondly, I want to thank the SocioCortex team as well as the team of the sebis chair for the great time and constructive conversations. Thank you to my friends and family who always supported me when I needed it.

And of course her, my motivation, who never gives up on me.

# Abstract

Hybrid Wikis try to combine the high applicability of ordinary wikis with the approach of semantic rich but static enterprise architecture modeling. The main concepts of these hybrid wikis are on the one hand keeping the loose structure of ordinary wikis, which means pages with pure text editable by users with specified rights and on the other side the combination of various meta data. There are for example attributes, type tags, attribute suggestions, as well as attribute definitions with integrity constraints.

Generating knowledge is often a process with loose structure. Therefore, an extra category called knowledge-intensive processes was introduced to support knowledge work. These processes normally cannot be expressed by typical business process management systems because of their unpredictability and error-proneness. Recent researches worked out requirements and characteristics which are needed to provide a system supporting knowledge-intensive processes including the ability for modeling and abstracting.

The objective of this master's thesis is to combine a Hybrid Wiki with aspects of a system supporting knowledge-intensive aspects. The focus is not to create an application for modeling purposes but to create an application, which helps to structure and document processes for knowledge work based on the idea of tasks combined with the characteristics of the Hybrid Wiki.

# Contents

# List of Figures

# Listings

# List of Abbreviations

| | |
|---|---|
| **ACE** | Attempto Controlled English |
| **AIFB** | Institute of Applied Informatics and Formal Description Methods |
| **API** | Application Programming Interface |
| **ASCII** | American Standard Code for Information Interchange |
| **BPMN** | Business Process Modeling Notation |
| **CSS** | Cascading Style Sheets |
| **CMMN** | Case Management Modeling Notation |
| **CRUD** | Create Read Update Delete |
| **HTML** | Hypertext Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **KIP** | knowledge intensive process |
| **IT** | Information technology |
| **JS** | JavaScript |
| **JSON** | JavaScript Object Notation |
| **MxL** | model-based expression language |
| **MVC** | Model View Controller |
| **NLP** | Natural Language Processing |
| **NPM** | NodeJS Package Manager |
| **OMG** | Object Management Group |
| **ORM** | Object-Relational Mapping |

| | |
|---|---|
| **OWL** | Web Ontology Language |
| **PDF** | Portable Document Format |
| **REST** | Representational State Transfer |
| **RDF** | Resource Description Framework |
| **RDFS** | Resource Description Framework Schema |
| **SASS** | Syntactically Awesome Style Sheets |
| **SCSS** | Sassy CSS |
| **sebis** | Software Engineering for Business Information Systems |
| **SPARQL** | SPARQL Protocol and RDF Query Language |
| **SMW** | Semantic MediaWiki |
| **SMW$^{+}$** | Semantic Enterprise Wiki |
| **SVG** | Scalable Vector Graphics |
| **UI** | User Interface |
| **URL** | Uniform Resource Locator |
| **UTF-8** | 8-Bit UCS Transformation Format |
| **WWW** | World Wide Web |
| **W3C** | World Wide Web Consortium |
| **WYSIWYG** | What You See Is What You Get |

# Outline of the Thesis

## Part I: Introduction

CHAPTER 1: INTRODUCTION & MOTIVATION

This chapter presents a short overview for the current approach of SocioCortex and briefly explains how to extend SocioCortex to provide a task-centric social content management application.

CHAPTER 2: PREVIOUS CONCEPTS OF SOCIOCORTEX

SocioCortex is based on an application called Hybrid Wiki. The task-related concepts are related to an application called Darwin Wiki. This chapter compares these two approaches to find differences.

CHAPTER 3: RELATED WORK

To provide a bigger scope related to knowledge generation and structure finding this chapter discusses other approaches of this area.

CHAPTER 4: USED TECHNOLOGIES AND LIBRARIES

The implemented solution is based on existing techniques and methodologies which are explained in this chapter.

## Part II: Contribution

CHAPTER 5: GENERIC CLIENT FOR SOCIOCORTEX

This chapter presents the use cases which are going to be supported and the basic structure of the SocioCortex Generic Client

CHAPTER 6: TASK SUPPORT IN SOCIOCORTEX

Based on the use cases of chapter 5, this chapter develops the conceptual model needed for supporting these use cases. It also explains further implemented concepts regarding SocioCortex.

## Part III: Conclusion

CHAPTER 7: SUMMARY & CONCLUSION
This final chapter summarizes the reached objectives and provides an outlook for further improvements of the system.

# Part I.

# Introduction

# 1. Introduction & Motivation

Wikis are tools for managing knowledge and websites. They empower people to contribute in gaining knowledge and are enabling to distribute the knowledge. A wiki's openness is allowing everyone to share knowledge. This is one of the most important parts of a wiki [SBBK07]. Another important aspect is flexibility for users. It allows different working methods without defining the used technologies. However, access and flexibility of data can lead to proliferation, which means to add too many data so they can not be moderated or reviewed anymore [SBBK07, HT08]. However, Wikipedia, according to Alexa.com, is the seventh most visited website in the world, it seems not to suffer from its huge range and enormous number of visitors [HT08]. Whereas, enterprise wikis seem to be influenced by proliferation due to their lack of pre-defined structures [HT08].

The Hybrid Wiki is an approach to facilitate emergent adaptive information structures in enterprise wikis. Using a free-form nature the Hybrid Wiki allows to create and persist knowledge and informations. This enables to form a structure around it using continuous adaptions [Neu12]. Therefore, the goal of the Hybrid Wiki is to empower end-users to easily enrich the unstructured content of a wiki page with structured information. Using these informations allows model experts to derive domain specific models which then can be used as templates for new content.

A successor of the Hybrid Wiki is called Darwin Wiki. In contrast to the Hybrid Wiki, it is not focusing on data and its structures. The focus is on creating and structuring processes. Therefore, wiki pages are regarded as the result of a process. Each process can be divided in different single tasks which are expecting results in form of structured data. The goal is to empower end-users to structure processes collaboratively [HKM15].

## 1.1. Motivation

SocioCortex is another successor of the Hybrid Wiki. The difference of SocioCortex' Core Application is the approach of not being a complete system than only providing a REST API for the purposes the Hybrid Wiki is intended for. Similar to the Hybrid Wiki the main purpose of SocioCortex is to support the collaborative creation of knowledge. Due to the fact that knowledge creation is a process SocioCortex is lacking the capabilities of the Darwin Wiki for modeling knowledge-intensive processes [CL02].

Additionally, the extensive use of a REST API enables SocioCortex to offer a variety of clients. Every client is dedicated to a special purpose. For example, there is a client which facilitates the creation of domain specific models whereas another client is used for visualizing the data which are persisted in these models. A third client is targeting the problem for mass editing single instantiated entities of the provided model.

## 1.2. Problem Statement

Processes are one part of knowledge generation. As SocioCortex aims at supporting knowledge generation the need to support processes is created. However, supporting these was not intended by the Hybrid Wiki. Therefore, SocioCortex states the requirement for being a collaborative information system and to support knowledge generation which creates the need for enabling a task-centric environment [seb14]. The relation of the Darwin Wiki to SocioCortex opens up the possibility for extending SocioCortex with concepts of the Darwin Wiki regarding knowledge intensive processes (KIPs).

Empowering end-users in creating knowledge, data structures and processes is one of the most important goals of the Hybrid Wiki and the Darwin Wiki. Regarding the current clients of SocioCortex none of these is supporting end-users in contributing to knowledge and process generation.

## 1.3. Objectives

This thesis provides an initial approach for modeling processes and their execution in SocioCortex. To achieve this the conceptual model of SocioCortex is going to be extended using the results gained from the Darwin Wiki. A second objective is the creation of a basic SocioCortex client for supporting end-users with basic task management and the ability in contributing to knowledge generation.

At first the Hybrid Wiki and Darwin Wiki are compared considering the different focuses of the applications. The detailed explanations will be extended by a short overview of related work regarding other approaches for generating knowledge.

Secondly, the new client for end-users is introduced. The client is also focusing features presented by the Darwin Wiki. Therefore, use cases are stated which are providing the scope of the created client.

Regarding the use cases an extension for the already existing model of SocioCortex is provided. The priority is to maintain existing functionalities with solely adding new capabilities. Therefore, the conceptual models of the Hybrid Wiki and the Darwin Wiki are compared. The gained information is then used to implement this concept to SocioCortex.

SocioCortex is no regular standard software. Providing a short methodology for adding new capabilities is also an objective of this thesis. The focus of SocioCortex on its REST API is used to limit the explanations to the persistence and REST layer.

# 2. Previous Concepts of SocioCortex

SocioCortex's goal is to combine and develop Tricia's approach of being a Hybrid Wiki and Darwin's support for knowledge intensive work [seb14].

Before presenting the resulting conceptual model as well as the approach to achieve this in chapter 5, this chapter explains the important parts of Tricia and Darwin to provide better understanding for the decision made in the following chapters.

First it is explained how end-users can participate in evolving the models within a Hybrid Wiki. The Darwin Wiki is able to handle loosely structured to even unstructured processes. Then it is described how the concepts of the Darwin Wiki are applied on these loosely structured data which creates the ability of developing whole processes and process structures out of them.

## 2.1. Tricia as a supporting tool for unstructured domains

Hybrid Wikis are basically a classical wiki application extended by features from semantic wikis. This means that features like attributes or categorization of content in terms of type tags enables gathering structured information of unstructured content [MNS11,BSVW12].

The data model for Hybrid Wikis is shown in figure 2.1. It mainly consists of two hierarchical structures. The tree on the right can be seen as a meta layer for the data of the wiki pages. It provides conditions, restrictions and guidance for the second tree, the instance layer.

Data persisted in the left tree can be seen as the wiki pages itself. For example, the entity `WikiPage` contains the unstructured content of a wiki page as well as other metadata of the page like title or author. The Entity `Attributes` provides another example for the left tree being the instance layer. It contains additional data for every wiki page like explained in section 2.1.2.

Primarily two additions were made to provide the structural data in the Hybrid Wiki [MNS11]: type tags and attributes.

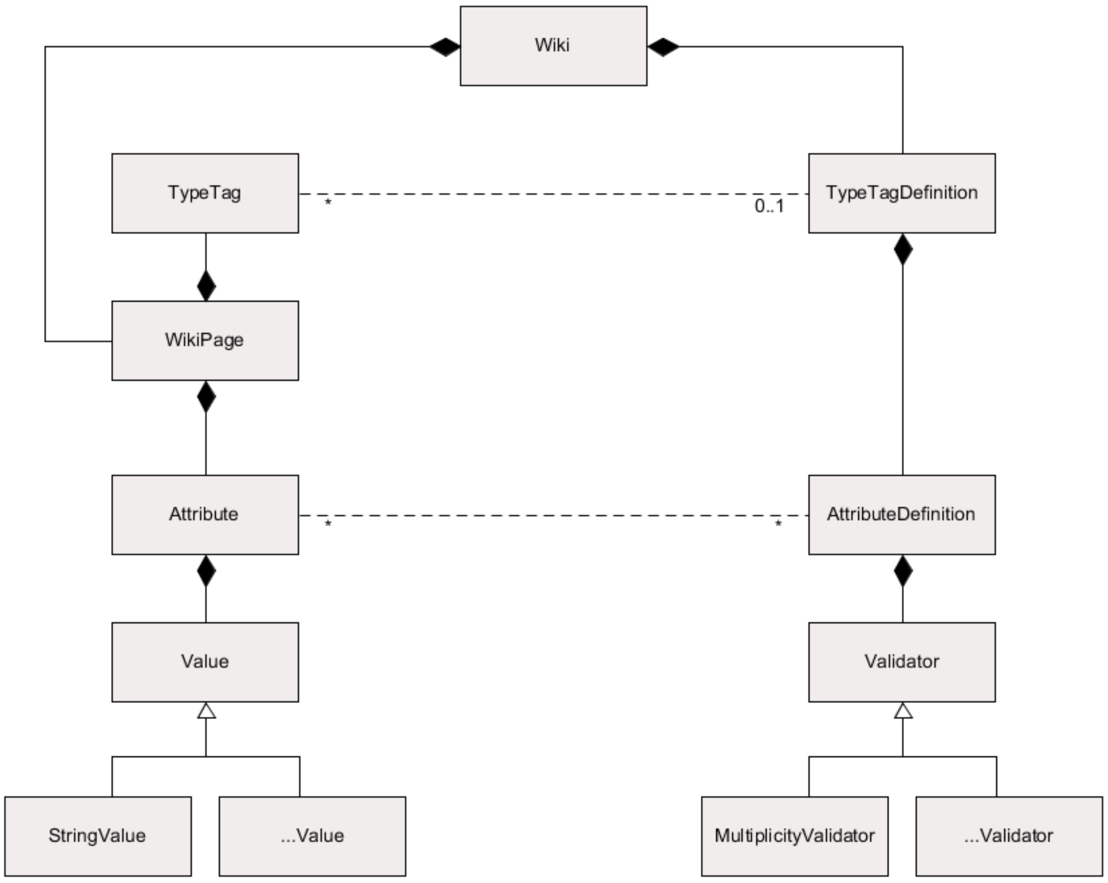


Figure 2.1.: The adapted conceptual model of the Hybrid Wiki prepared for the comparison of the Hybrid Wiki vs. the Darwin Wiki. Adapted from [MNS11]. The original model can be found in figure 8.1

### 2.1.1. Type Tags

In classical wikis, tags in general are used to categorize pages and their content. Every page of the wiki can be assigned to multiple tags, generally consisting of keywords of the text from the page itself [MNS11].

At the beginning of a new Hybrid Wiki instance, type tags are simple labels for pages used to give a rough categorization. Later on, type tags develop to very flexible templates for wiki pages by being the connection from the instance layer to the type layer [MNS11]. That is, type tags are used by the user to indicate what the content is about and enable the system to recommend which structural data in form of attributes could be added to the page [MNS11].

### 2.1.2. Attributes in Tricia

Attributes are the structured part of a Hybrid Wiki's page compared to the unstructured content of classical wiki pages. Basically attributes are key-value pairs added by the user to add additional content information.

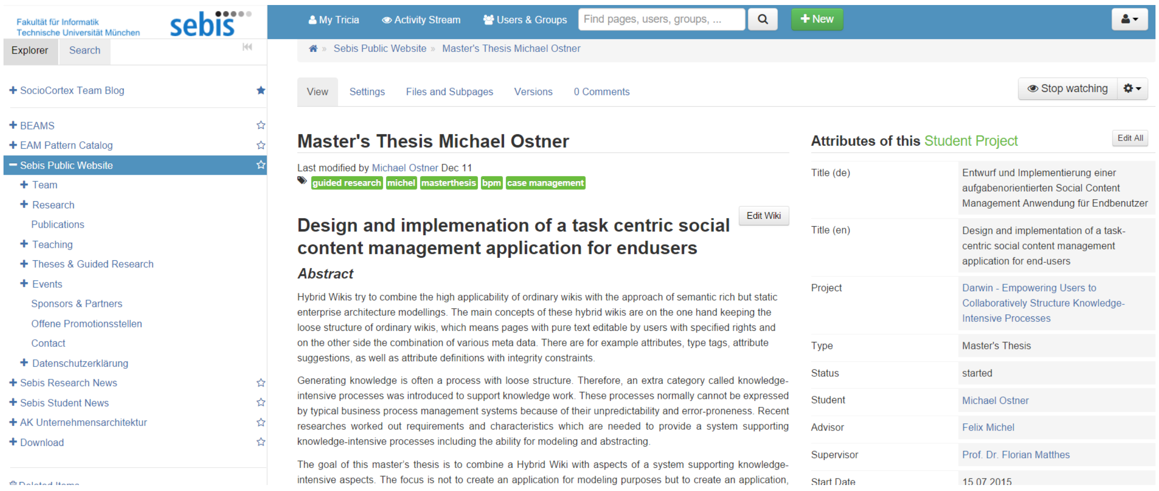
The feature is realized with a box on the right side of Tricia's content pages. In figure 2.2 one these boxes are shown. On top of the box the type of the current page is mentioned. It is also used as a link to the type's page where more information about the type can be seen as well as hyperlinks to all other instances of this type.

Within this box the attributes are listed in a table whereas the attribute's name is printed on the left and its value is printed on the right side. Even though the attributes' keys are always simple strings, whereas values can consist of different data forms. Some of the allowed value types besides plain text values are integers, floats, a selection of fix values, or even cross-references to other entities. Furthermore, an attribute's value has the ability of having different multiplicities. This means that an attribute can have one, none or multiple values of different types.

In figure 2.2 the attributes `Title(de)` and `Title(en)` are of type text. Other attributes like `Project` and `Student` are cross-references to other entities. `Project` in this case represents the underlying project and by `Student` a reference to the processing student is made.

Editing attributes is available by either hovering the value of an attribute and clicking on the appearing button labelled with "Edit". The value is then replaced by an editable field which helps the user by taking the value's type into account, too. For example, dates can be chosen with a date-picker and references can be made by an auto-completed list.

A second possibility changing the values of attributes is illustrated in figure 2.2. If the user is having write access to the page a button is enabled at the right of the type's name. The button has the lettering "Edit All". Clicking on this button switches all fields of the page into edit mode which enables changing the values all at once.



**Master's Thesis Michael Ostner**

Last modified by Michael Ostner Dec 11

**Design and implemenation of a task centric social content management application for endusers**

***Abstract***

Hybrid Wikis try to combine the high applicability of ordinary wikis with the approach of semantic rich but static enterprise architecture modellings. The main concepts of these hybrid wikis are on the one hand keeping the loose structure of ordinary wikis, which means pages with pure text editable by users with specified rights and on the other side the combination of various meta data. There are for example attributes, type tags, attribute suggestions, as well as attribute definitions with integrity constraints.

Generating knowledge is often a process with loose structure. Therefore, an extra category called knowledge-intensive processes was introduced to support knowledge work. These processes normally cannot be expressed by typical business process management systems because of their unpredictability and error-proneness. Recent researches worked out requirements and characteristics which are needed to provide a system supporting knowledge-intensive processes including the ability for modeling and abstracting.

The goal of this master's thesis is to combine a Hybrid Wiki with aspects of a system supporting knowledge-intensive aspects. The focus is not to create an application for modeling purposes but to create an application,

**Attributes of this Student Project**

| | |
|---|---|
| Title (de) | Entwurf und Implementierung einer aufgabenorientierten Social Content Management Anwendung für Endbenutzer |
| Title (en) | Design and implementation of a task-centric social content management application for end-users |
| Project | Darwin - Empowering Users to Collaboratively Structure Knowledge-Intensive Processes |
| Type | Master's Thesis |
| Status | started |
| Student | Michael Ostner |
| Advisor | Felix Michel |
| Supervisor | Prof. Dr. Florian Matthes |
| Start Date | 15.07.2015 |

Figure 2.2.: Tricia's representation of a wiki page

### 2.1.3. Evolving models in Tricia

Type tags are a first approach to structure plain textual content. Assigning type tags to wiki pages enables to search in pages with the same type tag for common properties. Detected common properties can then be lifted to the type layer [MNS11].

The majority of these properties are attributes on wiki pages. If a relative high number of wiki pages have the same attributes, a user with sufficient rights will be able to create an attribute definition out of these attributes. Same attributes in this context means that these attributes have the same key and not necessarily the same values.

Attribute definitions have the possibility to various constraints to the values of the attributes they are assigned to. For example, there are multiplicity constraints, type constraints for values and additional type specific constraints like ranges for numbers.

On the one hand attribute definitions are restrictions for attributes but on the other hand they are also providing guidance for end-users to know which value is expected in a specific attribute.

These kind of structured data empowers the Hybrid Wiki to compare pages using quantitative measurements. Mentioned above the values of attributes don't have to be plain text. Using numbers or values similar to numbers like dates enable the user to compare the different wiki pages on absolute values [MNS11].

The attributes are coupled to the attributes definition by name only. This loose connection allows a degree of freedom, which is used to for further steps of evolving. The coupling of attributes and attribute definitions can easily be changed. Furthermore, attributes can still be created without a definition and added to the page because every page has its own list of attributes. [MNS11].

Using these mechanics over time the Hybrid Wiki is able create a model of a specific domain which is evaluated and evolved by the end-users.

## 2.2. Darwin as a supporting tool for unstructured processes

The Business Process Modeling Notation (BPMN) was developed in May 2004. The goal was to create a modeling language for processes which was understandable by all business users who is participating in the process [Whi04]. Unfortunately, classic workflow management seems to be too restricted in order to use it for knowledge workers and KIPs [VSW03]. Therefore, in 2001 case management was introduced to support knowledge workers dealing with KIPs. In workflow management, which uses predefined process control structures, e. g. modeled with BPMN, it defines what should be done during a workflow process whereas case handling focuses on what can be done to achieve a business goal [RRV03].

In 2014, the Case Management Modeling Notation (CMMN) was presented by the Object Management Group (OMG). Rather than describing the process in an imperative way its goal is to provide a descriptive view on the process [Obj14, MHV13]. Specifying what should be done without specifying of how it should be done creates the needed flexibility to deal with unpredictable events [PV06].

There are attempts to create subsets of the CMMN because it seems to be too complex. Within these requests it is stated that the goal should be to make the notation more accessible to end-users without a computer science background as well as people who are not in the need to understand the whole CMMN [MLV14, HKM15]. Therefore, Darwin's primary goal is to make the structuring of knowledge-intensive processes easier, so that non-experts without knowledge about a dedicated process notation can contribute to the modeling process [HKM15].

Darwin is a kind of a wiki, too because it uses the experiences made by the Hybrid Wiki and is based on some of its concepts. Therefore, it supports similar features with having a different focus [HKM15].

Two similar core concepts are attributes and types. Tasks are newly introduced as the third core concept used in the Darwin Wiki in order to empower the end-user to create process structures while doing the process itself.

### 2.2.1. Similarities to Tricia

The relation between the Hybrid Wiki and the Darwin Wiki can be observed by comparing the conceptual data models of these two. Figure 2.1 shows the Hybrid Wikis model, whereas figure 2.3 shows the model for the Darwin Wiki. For easier comparing these two are adapted versions of the original models attached in chapter 8 on page 75.

Both models have two main trees representing the type layer and the instance layer. This indicates that the resulting features are quite similar. The `Attributes` of a newly created instance of a `WikiPage` will be initiated based on the `AttributeDefinitions` of the according

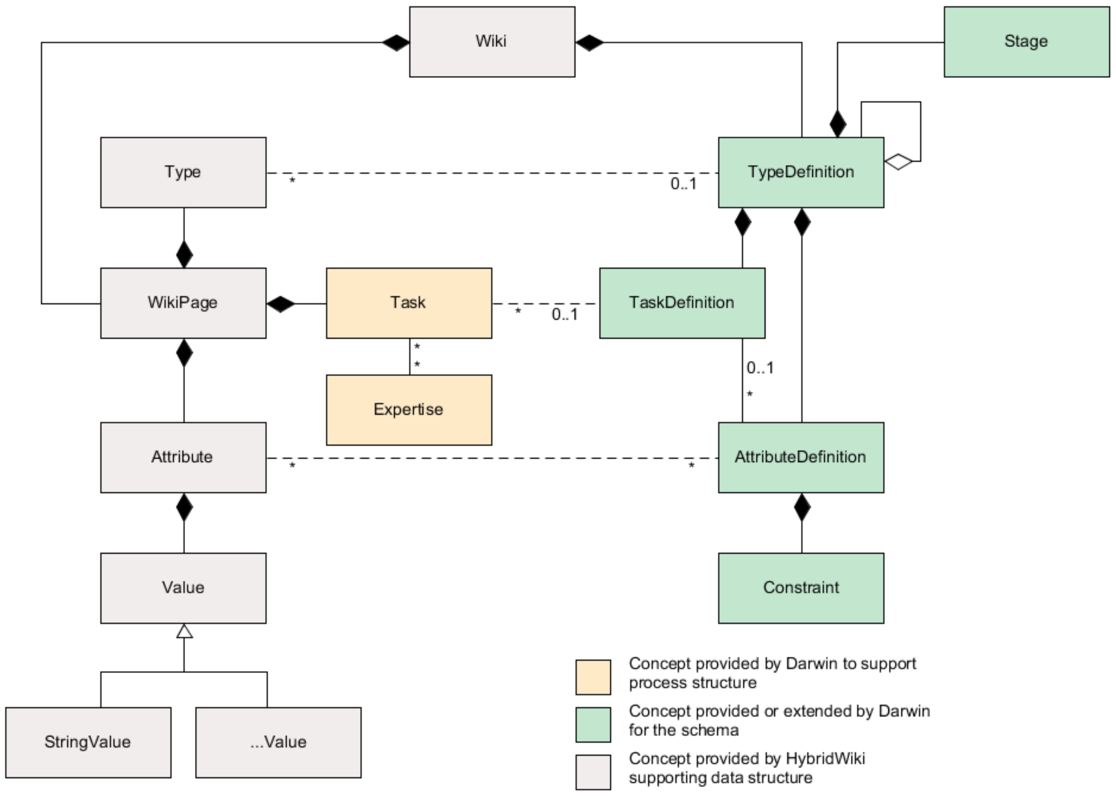
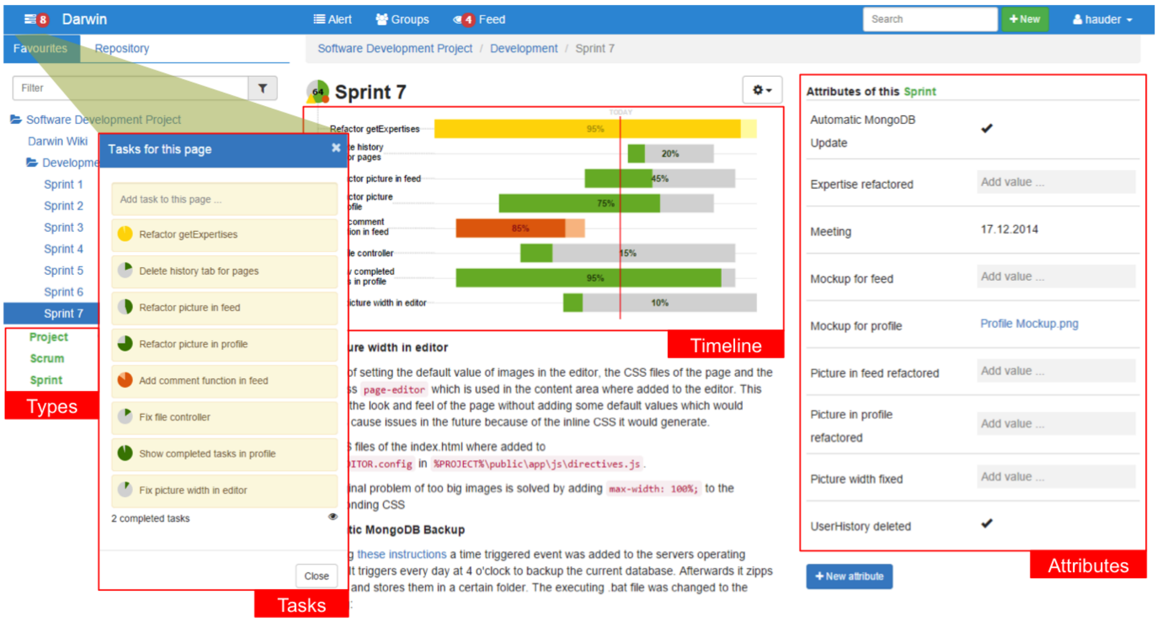

Wiki
Stage
Type
TypeDefinition
*
0..1
WikiPage
Task
TaskDefinition
*
0..1
*
*
Expertise
0..1
*
Attribute
AttributeDefinition
*
*
Value
Constraint
StringValue
...Value
Concept provided by Darwin to support process structure
Concept provided or extended by Darwin for the schema
Concept provided by HybridWiki supporting data structure

Figure 2.3.: The adapted conceptual model of the Darwin Wiki prepared for the comparison of the Hybrid Wiki vs. the Darwin Wiki [HKM15]. The original model can be found in figure 8.2

`TypeDefinition`. The realization is illustrated in figure 2.4. Compared to the screenshot of the Hybrid Wiki it has a very similar structure, too. As explained the box on the right side is containing attributes like described in section 2.1.2. The type tag is placed above the attributes' box. Editing the values is also very similar to Tricia by clicking on an edit button.



ure width in editor

of setting the default value of images in the editor, the CSS files of the page and the ss `page-editor` which is used in the content area where added to the editor. This the look and feel of the page without adding some default values which would cause issues in the future because of the inline CSS it would generate.

files of the index.html where added to `ITOR.config` in `%PROJECT%\public\app\js\directives.js`.

inal problem of too big images is solved by adding `max-width: 100%;` to the nding CSS

tic MongoDB Backup

g these instructions a time triggered event was added to the servers operating It triggers every day at 4 o'clock to backup the current database. Afterwards it zipps and stores them in a certain folder. The executing .bat file was changed to the

Figure 2.4.: Screenshot of the Darwin Wiki [HKM15]

### 2.2.2. Task Concept of Darwin

The biggest difference between the Hybrid Wiki and the Darwin Wiki is the implementation of tasks and process support. Regarding the model of figure 2.3 in Darwin a process is structured with tasks and stages. Similar to attributes every instance of a `WikiPage` can have multiple attached instances of `Tasks`. Furthermore, tasks can have a definition which will be used to initiate and predefine tasks when a page is initiated [HKM15].

The progress' state of a page is determined by taking the average of all progress of the tasks of the page. Basically this implies that every page can be seen as a task which has subtasks. The tasks of a page would then be a decomposition of this page [HKM15].

Additionally, tasks can have assigned attributes which are a subset of the page's attributes. In this case attributes are deliverables which should be fulfilled to let the task proceed in its progress. The implied hierarchical structure stated above is able to be achieved by adding a new page to the attributes of a task.

For example, there is a navigation tree on the left side of figure 2.4. `Development` is the parent of the currently chosen `Sprint7`. `Development`'s progress is calculated by taking the average progress of `Sprint1` to `Sprint7` because they are all values of attributes of tasks in `Development`.

Interesting for the end-user's point of view is the highlighted task listing on the left of figure 2.4. This list provides an overview for all tasks of a page as well as their progress status in form of an attached pie chart. Tasks can be created on the fly by typing a name in the list view and pressing enter.

Metadata are containing informations about a task like the date of the beginning and the date of the anticipated ending. To assign metadata the user has the opportunity to click on a task presented in the side bar. This opens its metadata editing mode. Metadata can be assigned for tasks derived from a task's definition as well as for tasks which are created using the feature of the side bar. Darwin uses the following metadata:

**start date**
determines at which date the task should be started

**end date**
reflects when the task should end

**progress**
is calculated automatically using the attributes unless the user wants to enter an own value then automatic calculation will be turned off and the users value will be persisted

**expertises**
describe which skills are necessary to be able to finish the task

**delegated to**
a person which is then in charge to finish the task

The center of the page is displaying a timeline similar to a Gantt chart. It represents the relative timely positioning of the tasks and helps to indicate the priority of a task. The positioning is calculated with the begin and end dates of the metadata.

The color of the bars are the same colors used for the according pie chart. They are intending the status of a task. Yellow implies an inconsistent state. An example for an inconsistent state is to have begin and end dates which are not in between the date range provided by the parent task. Red on the other hand implies a task which is overdue. Overdue tasks are having an end date which is past the current date.

### 2.2.3. Process Support of Darwin

The processes are designed and created by the end-users using similar techniques as described in section 2.1.3. A user starts working by creating pages and assigning attributes as well as tasks. During the process execution a modeling expert is able to evaluate the inserted data and to adapt the templates and constraints by the type layer to improve the experience for the end-user [HKM15].

Besides the possibilities of creating definitions the modeling expert has the possibility to create processes using a CMMN-Editor. The processes created by this editor can have several stages or constraints e. g. regarding their execution order. The execution order is typical constraint to force the user for completing a task before being able to execute a following task.

# 3. Related Work

SocioCortex is based on two systems, the Hybrid Wiki and Darwin Wiki [seb14]. This chapter is showing other types of wikis. Starting with the basic approach of the Wikipedia project in section 3.1, followed by an approach called semantic wikis in section 3.3 and other approaches which are relating informations in order to generate knowledge.

## 3.1. MediaWiki

The Wikipedia project may be the most known instance of the MediaWiki. First it was the server side software running Wikipedia. With growing success of Wikipedia the WikiMedia Foundation decided to launch a new project which was called MediaWiki and it was directly published as version 1.1 in 2003. Since this event Wikipedia only was an instance of the new MediaWiki engine which is made available by an open source licensing model. Therefore, MediaWiki is the classical implementation of a wiki software [Wik08].

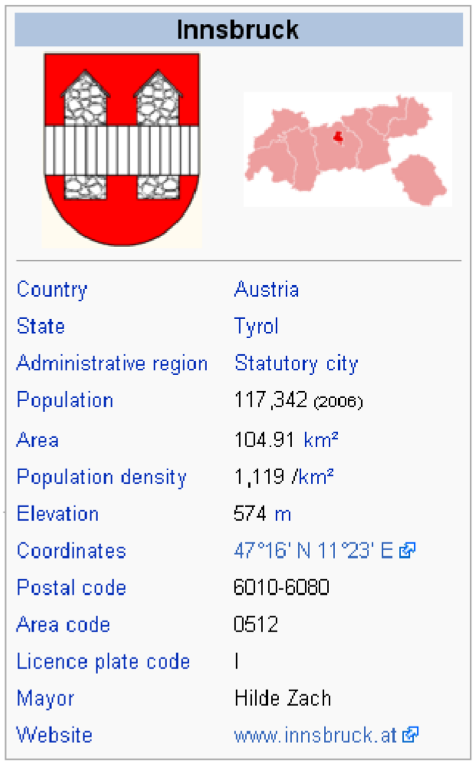
Wikipedia was at first only consisting of unstructured content, which means text only. Finding fast information was difficult because the user was forced to read the text in order to find the searched information. Consequently, a need for structured content was growing [HLS05]. The MediaWiki's solution for this structuring problem are wiki templates which can be associated to a wiki page when the page is created [HLS05]. A sample of a template can be seen in figure 3.1.

Besides helping users to get the desired information, another benefit of templates is reusability. Users are able to use existing structures to fill in wanted information which gives guidance for the creation of content. They provide information of what kind of content is expected for the new wiki page. Hence, templates can be seen as a kind of categorization of the content [Neu12].

Nevertheless, templates have some downsides, too. Changing a template of an existing page is not as easy as creating a new instance of the template. They are created by mixing wiki specific markup, like the left part of figure 3.1, with markup elements similar to Hypertext Markup Language (HTML) which does the end-user require to have capabilities in web programming. Other negative aspects are that users cannot create new attributes, which are independent of a template, as well as the user does not know if values will still exist if the template or the template's structure changes. This makes the process of creating new structures or adopting old ones far more complex because the user cannot adopt changing requirements [Neu12].

The Hybrid Wiki's approach does not give strict structures like templates but it offers another way of creating structured content and typification by type tags, attributes and constraints, which will be explained in detail in chapter 2.

```
{{Infobox Town AT |
  name = Innsbruck |
  image_coa =  InnsbruckWappen.png |
  image_map = Karte-tirol-I.png |
  state = [[Tyrol]] |
  regbzk = [[Statutory city]] |
  population = 117,342 |
  population_as_of = 2006 |
  pop_dens = 1,119 |
  area = 104.91 |
  elevation = 574 |
  lat_deg = 47 |
  lat_min = 16 |
  lat_hem = N |
  lon_deg = 11 |
  lon_min = 23 |
  lon_hem = E |
  postal_code = 6010-6080 |
  area_code = 0512 |
  licence = I |
  mayor = Hilde Zach |
  website = [http://innsbruck.at] |
}}
```



**Innsbruck**

| | |
|---|---|
| Country | Austria |
| State | Tyrol |
| Administrative region | Statutory city |
| Population | 117,342 (2006) |
| Area | 104.91 km² |
| Population density | 1,119 /km² |
| Elevation | 574 m |
| Coordinates | 47°16' N 11°23' E |
| Postal code | 6010-6080 |
| Area code | 0512 |
| Licence plate code | I |
| Mayor | Hilde Zach |
| Website | www.innsbruck.at |

Figure 3.1.: Example of a Wikipedia template with an infobox on the left and the rendered MediaWiki output on the right. This example is applied for Innsbruck, an Austrian town. [AL07].

## 3.2. Semantic Web

The Hybrid Wiki and therefore SocioCortex as well do not support the Semantic Web standard. However, it would be a possibility of enhancing SocioCortex to support it at least in public pages. Nevertheless, there are other wiki implementations presented in section 3.3 which are using these approaches.

The problem of the current World Wide Web (WWW) is heterogeneity made of the possibilities data can be presented. Starting with different encodings like ASCII and UTF-8 continuing with different representations like plain text or HTML and file formats like PDF [HKRS08].

The objective of the Semantic Web is to provide a standard for exchanging data in a consistent way and to connect informations in a semantically enriched way in order to generate implicit knowledge [BLHL01, HKRS08]. Therefore, the data must be processable for applications as well as for humans. Moreover, the standard has to be flexible and expandable [HKRS08]. For this purpose, the World Wide Web Consortium (W3C) introduced the Resource Description Framework (RDF) and the Web Ontology Language (OWL).

The main concept of RDF is to make resources addressable and connecting them by adding meanings to the relation between the resources. Hence, RDF encodes the information of a resource into a Graph Data Model. The graph is consisting of date triples which every of these has a subject, a predicate and an object [HPS14]. An object of one triple can be the subject of another triple as well as a predicate can be an entire resource. Figure 3.2 shows the example of a RDF graph model describing the relations of a written artifact.

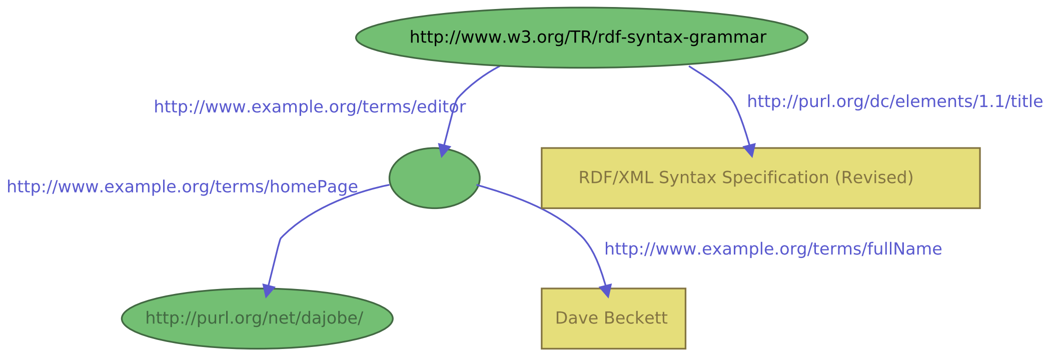

http://www.w3.org/TR/rdf-syntax-grammar
http://www.example.org/terms/editor
http://purl.org/dc/elements/1.1/title
RDF/XML Syntax Specification (Revised)
http://www.example.org/terms/homePage
http://www.example.org/terms/fullName
http://purl.org/net/dajobe/
Dave Beckett

Figure 3.2.: Graph for a RDF/XML Example [Bec04]. The subject `RDF grammar` has a predicate `title` which's values is a string. The second predicate describes the `editor` of the definition which is pointing to another resource. This resource has the predicates `fullName` and `homePage`. The first one is another string value, whereas the latter one is again a resource.

The Resource Description Framework Schema (RDFS) is, similar to RDF, another recommendation by the W3C. It extends the RDF vocabulary with semantic extras like resources which can be a class or a subclass or an instance of a class but these extras are too limited for using in the WWW [BG14, Neu12]. Therefore, a new concept was created called Web Ontology Language (OWL), which is an extension of the RDFS. OWL adds more vocabulary like disjunction of classes, cardinalities or enumerated classes [MV09]. Another part worth mentioning in the context of the semantic web is SPARQL Protocol and RDF Query Language (SPARQL) which allows querying on RDF graphs similar to SQL in a relational database [PS08].

## 3.3. Semantic Wikis

The term *Semantic Wiki* is used in a variety of different systems. In general, they are connecting concepts of the semantic web of section 3.2 with the classical wiki approach of section 3.1 [SBBK07]. The problem they state is to find certain informations or facts in a wiki without reading a whole article if not multiple articles for finding one single information as well as it is difficult to find informations having constraints [KVV05]. Therefore, the concept of semantically linked articles using the RDF is proposed [KVV05]. Traversing through articles is then possible by seeing the current wiki page as the subject. A link to another resource or an annotation would represent the predicate and the targeting page would be the object. The URL of the used resources could serve as identifier. For example, in regular wikis the process of finding *James Bond movies from the 60s that were not starring Sean Connery* requires to search in different articles to find the desired information but using the semantic techniques could result in a graph illustrated in figure 3.3 [KVV05]. A small selection of semantic wikis is presented in the following sections.

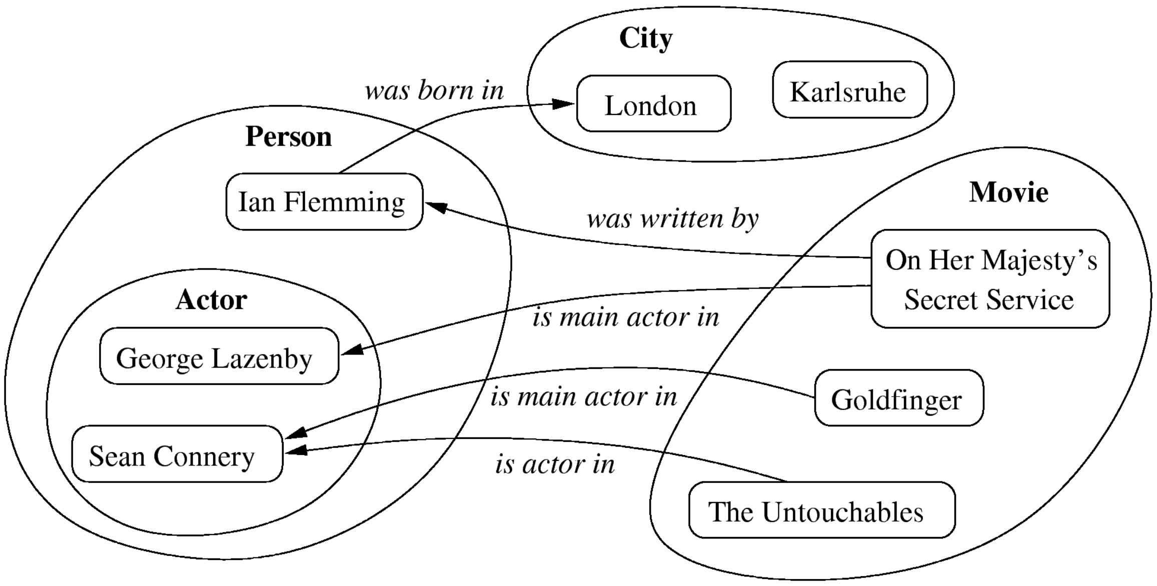


Figure 3.3.: Example of a RDF graph. The rounded rectangles are resources whereas the relations are displayed by directed arrows titled with the predicate name. They are always pointing from a subject to an object. The circles around the resources are classifications which are an extension of RDF by RDFS [KVV05]

### 3.3.1. Semantic MediaWiki

The Semantic MediaWiki (SMW) is basically an extension of the existing MediaWiki mentioned in section 3.1 [KVV06]. The system tries to address three core problems [KVV06]:

**Consistency of content**
is an issue because all data are edited manually and no automated updates can be made.

**Accessing knowledge**
: is difficult because the amount of pages in wikis, which makes comparing resources and finding related articles time-consuming.

**Reusing knowledge**
: is only possible by using a browser-like application, which is an obstacle in terms of sharing the knowledge by using other applications.

Beside the three core problems the main focus of the Semantic MediaWiki (SMW) is the seamless integration of semantic technologies into the MediaWiki and keeping the well-known usage patterns in order to be accessible for non-expert users [KVV06]. The three core problems are addressed by different concepts. Through querying using a language like shown in figure 3.4 it is possible to link data and use them as variables so these stay consistent along the system [KVV06]. The second problem is managed by introducing different concepts in the combination of concepts of RDFS. Pages can get a category which can be the subcategory of another category. Informations which do not have a resource and can therefore not be used by RDF are implemented as attributes which take the attributes name as predicate and the value as object [KVV06]. Third reusing knowledge is facilitated by enabling querying from outside the wiki through SPARQL [KVV06].

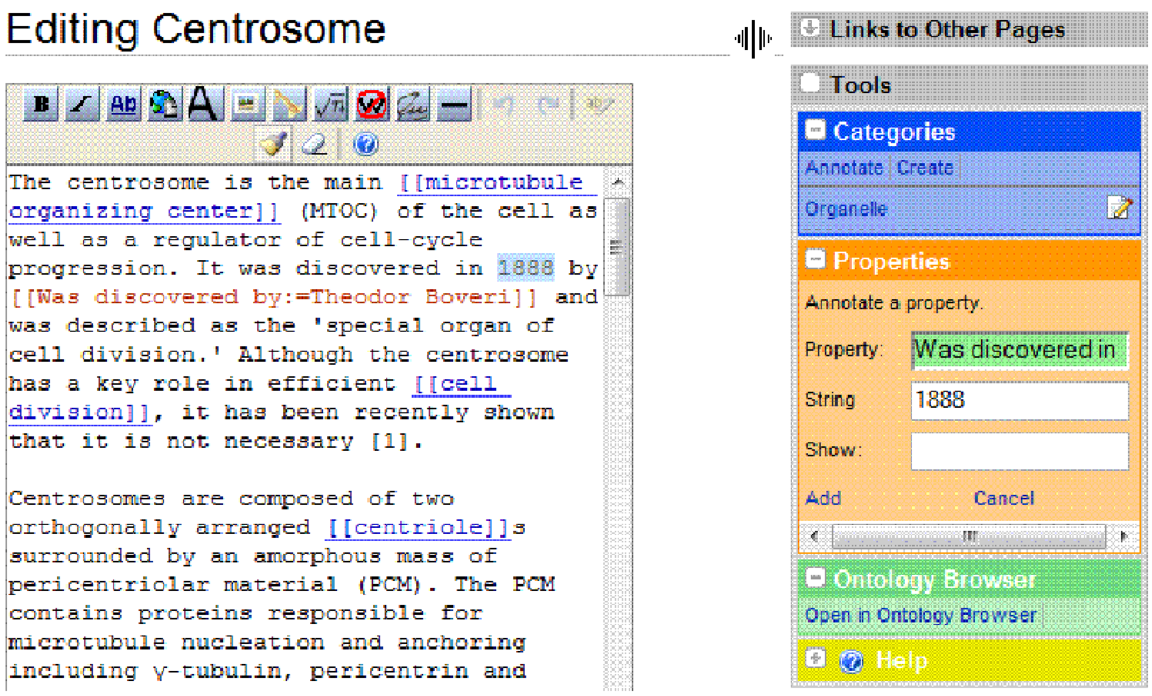
Hybrid Wiki and SocioCortex both have implemented a querying capability. They are using a model-based expression language (MxL) to access the data. Also these two wikis as well as Darwin have attributes and types and therefore, they are able to categorize knowledge and enhance the nonstructered data by structured informations. Furthermore, SocioCortex and Darwin can be used in other applications, too, because they are having a strict separation of the server components and the client using only a Representational State Transfer (REST) interface for accessing data.

```
[[Category:Conference]][[startdate:=>May142006]]
        [[programchair::<q>[[memberof::AIFB]]</q>]]
```

Figure 3.4.: A query using the syntax of the Semantic Media Wiki looking for instances of the category "Conference", which were starting after May 14th 2006 and had a program chair from Institute of Applied Informatics and Formal Description Methods (AIFB) of the Karlsruhe Institute of Technology [KVV06].

### 3.3.2. Semantic Enterprise Wiki

The Semantic Enterprise Wiki (SMW$^+$) is an extension of the SMW [Neu12]. The question it answers are how users which are not familiar to semantic technologies maximize their output of these technologies and how the users are able to get a maximum of help out of the semantic extension. Mainly a new interface component is introduced called Semantic-Toolbar [PNJB]. This toolbar provides an overview for attributes and other semantic data of the SMW like presented in figure 3.5. A second feature besides the toolbar is autocompletion for the text of the page. With this the user is made aware which predicates he can use to enrich the site's informations [PNJB].


Editing Centrosome

```
The centrosome is the main [[microtubule
organizing center]] (MTOC) of the cell as
well as a regulator of cell-cycle
progression. It was discovered in 1888 by
[[Was discovered by:=Theodor Boveri]] and
was described as the 'special organ of
cell division.' Although the centrosome
has a key role in efficient [[cell
division]], it has been recently shown
that it is not necessary [1].

Centrosomes are composed of two
orthogonally arranged [[centriole]]s
surrounded by an amorphous mass of
pericentriolar material (PCM). The PCM
contains proteins responsible for
microtubule nucleation and anchoring
including γ-tubulin, pericentrin and
```

Links to Other Pages

Tools

Categories

Annotate | Create

Organelle

Properties

Annotate a property.

| | |
|---|---|
| Property: | Was discovered in |
| String | 1888 |
| Show: | |

Add Cancel

Ontology Browser

Open in Ontology Browser

Help

Figure 3.5.: An example for the SemanticToolbar introduced by the Semantic Enterprise Wiki (SMW$^+$). On the left side there is the normal wiki markup extended by features of the Semantic MediaWiki (SMW). On right side the toolbar itself is placed, showing data which can be used in the wiki text. The data are proposed by a autocompletion feature.

### 3.3.3. AceWiki

Another approach for getting semantic informations out of unstructured data is to use Natural Language Processing (NLP). AceWiki is an approach which tries exactly this. It offers an editor which helps the user to create syntactically right sentences which can processed using Attempto Controlled English (ACE) [KS08]. The editor shown in figure 3.6 shows an example for an already started sentence. Studies proved that the autocompletion feature of the editor helps domain experts to create "expressive ontology languages" [Kuh09].

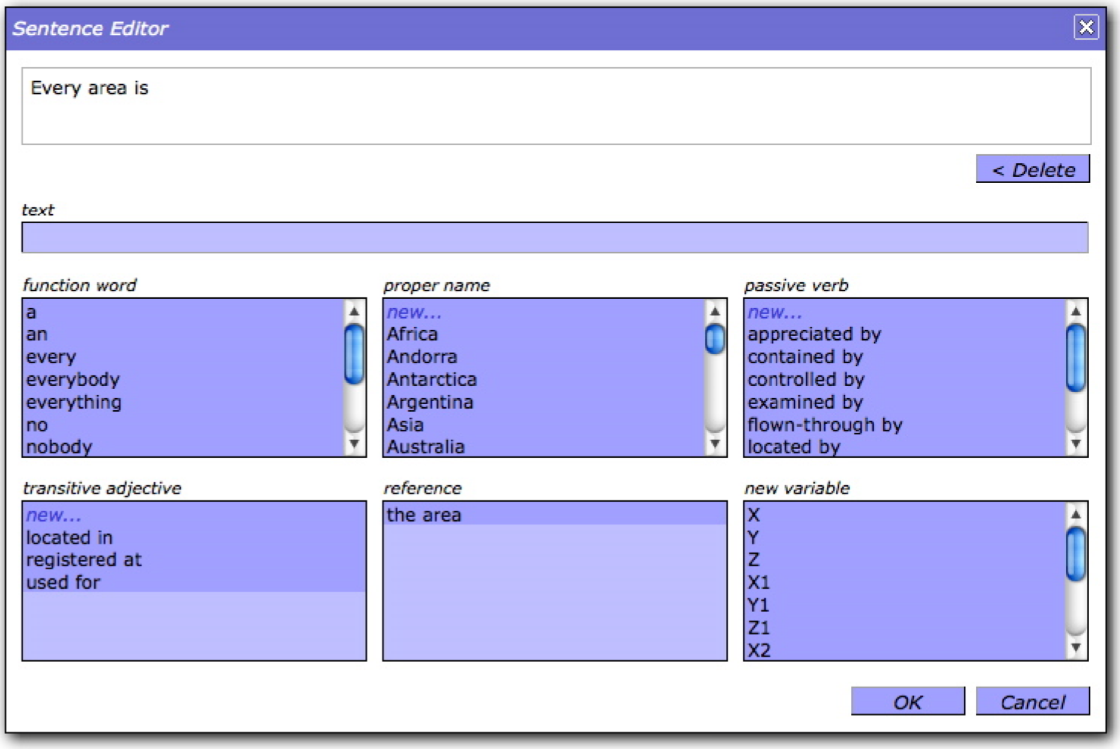
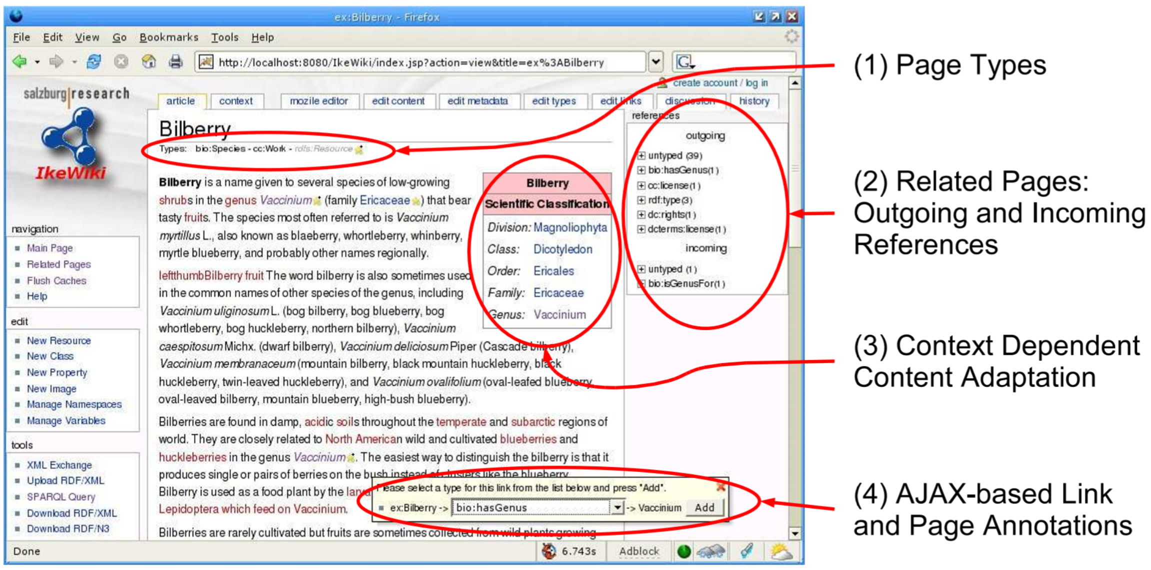


Figure 3.6.: The AceWiki editor helps in writing natural language processable sentences for automatic extraction of semantic data. The sentence is beginning with *Every area is* and the editor is proposing all possible partials to continue the sentence [KS08].

### 3.3.4. IkeWiki

IkeWiki was primarily developed as a tool for ontology engineering but it supports a variety of other features. One feature is the support for collaborative knowledge engineering and the support of different levels of formalization ranging from informal texts to formal ontologies [Sch06]. The wiki is facing the problem that domain experts are normally not having much technical expertise [Sch06]. Therefore, IkeWiki tries to keep the interface as close to the MediaWiki as possible in order to achieve a familiar look-and-feel experience to the user [Sch06]. The attempt to ease the user's experience is continued by adding a What You See Is What You Get (WYSIWYG) editor. For more experienced users IkeWiki offers using wiki markup[1] [Sch06].

IkeWikis architecture has different similarities with Tricia, Darwin and SocioCortex. All of these systems are persisting the unstructured data separate from the structured data. Whereas the wikis developed by the chair of Software Engineering for Business Information Systems (sebis) save the structured data in form of attributes with names and values, IkeWiki's approach is using a special RDF store, the Jena RDF framework. This allows fast in-memory querying of the data using SPARQL. Of course the memory data have to be persisted frequently in order to not get lost [Sch06].



Bilberry

Types: bio:Species - cc:Work - rdfs:Resource

**Bilberry** is a name given to several species of low-growing shrubs in the genus *Vaccinium* (family Ericaceae) that bear tasty fruits. The species most often referred to is *Vaccinium myrtillus* L., also known as blaeberry, whortleberry, whinberry, myrtle blueberry, and probably other names regionally.

leftthumbBilberry fruit The word bilberry is also sometimes used in the common names of other species of the genus, including *Vaccinium uliginosum* L. (bog bilberry, bog blueberry, bog whortleberry, bog huckleberry, northern bilberry), *Vaccinium caespitosum* Michx. (dwarf bilberry), *Vaccinium deliciosum* Piper (Cascade bilberry), *Vaccinium membranaceum* (mountain bilberry, black mountain huckleberry, black huckleberry, twin-leaved huckleberry), and *Vaccinium ovalifolium* (oval-leafed blueberry, oval-leaved bilberry, mountain blueberry, high-bush blueberry).

Bilberries are found in damp, acidic soils throughout the temperate and subarctic regions of world. They are closely related to North American wild and cultivated blueberries and huckleberries in the genus *Vaccinium*. The easiest way to distinguish the bilberry is that it produces single or pairs of berries on the bush instead of clusters like the blueberry.

Bilberry is used as a food plant by the larva Lepidoptera which feed on Vaccinium.

Bilberries are rarely cultivated but fruits are sometimes collected from wild plants growing

| Bilberry | |
|---|---|
| **Scientific Classification** | |
| *Division:* | Magnoliophyta |
| *Class:* | Dicotyledon |
| *Order:* | Ericales |
| *Family:* | Ericaceae |
| *Genus:* | Vaccinium |

Please select a type for this link from the list below and press "Add".

ex:Bilberry -> bio:hasGenus -> Vaccinium Add

Figure 3.7.: The interface of the IkeWiki, similar to Hybrid Wiki's tags there are page types under the page's name (1). The main content of the page is placed in the center. It is consisting of unstructured text. The right side contains attributes (3). The rightest space is reserved for related pages, which means incoming and outgoing references (2) [Sch06]. Not in Tricia implemented is the possibility of adding semantic data to the links in the unstructured content area.

[1] A short introduction to wiki markup: `http://www.mediawiki.org/wiki/Help:Formatting`, accessed on January 14th 2016

### 3.3.5. KiWi

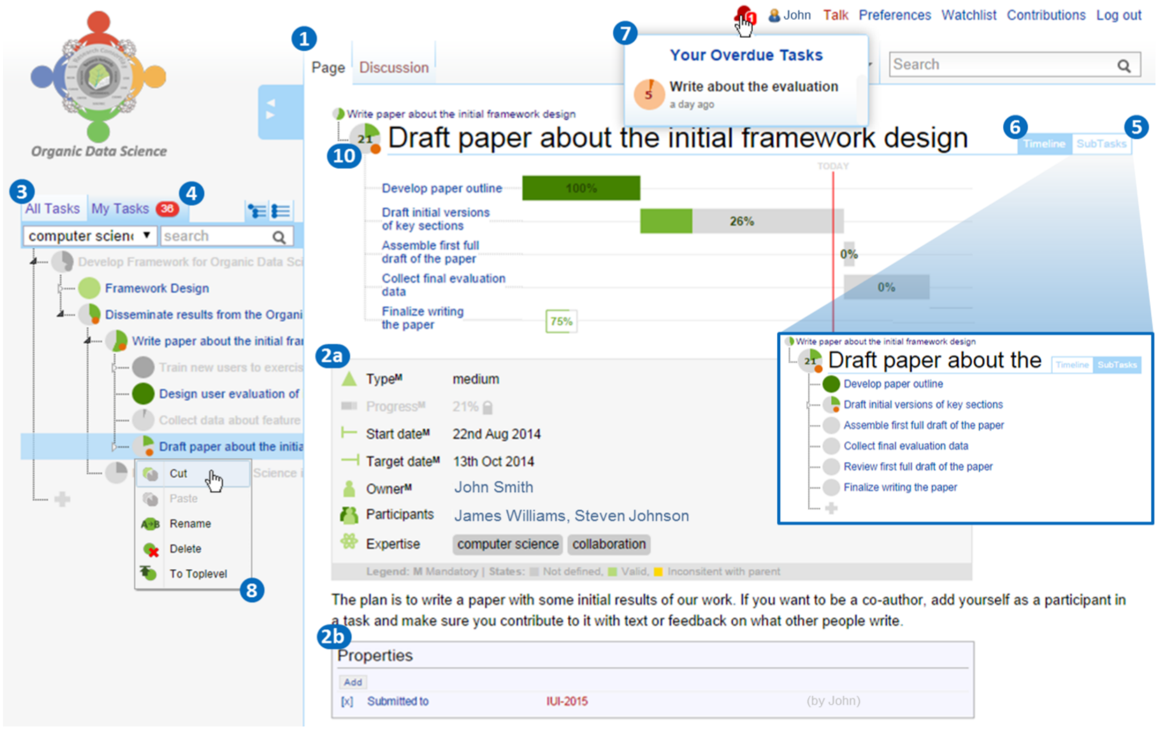
The eu-funded project KiWi is the successor of IkeWiki [Neu12]. It provides a similar look to IkeWiki like the center of the page containing unstructured content and the right side having attributes and references. The big difference between IkeWiki and KiWi is the decision of storing data in a semi-structured form [SSSF09]. Though in IkeWiki the structure of a wiki page in terms of type and category were strict and not able to be changed while creating or editing a wiki page, Kiwi now introduces type tags and tags, which are similar to the Hybrid Wiki's tags [Neu12]. The metadata of a page are presented in RDF tuples, which enables changing the model on runtime because RDF has no required schema definition [Neu12].

Though KiWi and Hybrid Wiki are having some capabilities in common, there are some differences, too. For example, Hybrid Wiki a wiki page and a blog post cannot represent the same content object because in Hybrid Wiki they are both individual entities [Neu12]. Another difference is the treatment of new entity types. In the Hybrid Wiki can only be created by changing the database to support a new entity representing the type [Neu12]. In KiWi the database does not have to be altered. Instead the new type has to be implemented by extending the data model in the application layer and by creating the corresponding views [Neu12].

## 3.4. Organic Data Science

Organic Data Science is a framework for computationally-grounded science collaborations. It addresses a problem for knowledge workers which are working in widely distributed teams. The solution is a task-centered framework which supports scientists across all levels [GMR$^+$15]. The implementation of the Organic Data Science Wiki is based on the Semantic Media Wiki. It is using its semantic capabilities for structuring content as well as for task-related properties [MGRH15].

The contribution of users is based on tasks which can be decomposed into smaller subtask. Like stated by RDF every task has its own page provided by an unique Uniform Resource Locator (URL). The page of a single task is illustrated in figure 3.8.


Organic Data Science
1
2a
2b
3
4
5
6
7
8
10
John Talk Preferences Watchlist Contributions Log out
Page Discussion
Search
Your Overdue Tasks
Write about the evaluation
a day ago
All Tasks My Tasks 36
computer scienc
search
Develop Framework for Organic Data Sci
Framework Design
Disseminate results from the Organi
Write paper about the initial fra
Train new users to exercis
Design user evaluation of
Collect data about feature
Draft paper about the initia
Science
Cut
Paste
Rename
Delete
To Toplevel
Write paper about the initial framework design
Draft paper about the initial framework design
Timeline
SubTasks
TODAY
Develop paper outline
100%
Draft initial versions of key sections
26%
Assemble first full draft of the paper
0%
Collect final evaluation data
0%
Finalize writing the paper
75%
Draft paper about the
Develop paper outline
Draft initial versions of key sections
Assemble first full draft of the paper
Collect final evaluation data
Review first full draft of the paper
Finalize writing the paper
Legend: M Mandatory | States: Not defined, Valid, Inconsistent with parent
Properties
Add

| | |
|---|---|
| Type$^M$ | medium |
| Progress$^M$ | 21% |
| Start date$^M$ | 22nd Aug 2014 |
| Target date$^M$ | 13th Oct 2014 |
| Owner$^M$ | John Smith |
| Participants | James Williams, Steven Johnson |
| Expertise | computer science collaboration |

The plan is to write a paper with some initial results of our work. If you want to be a co-author, add yourself as a participant in a task and make sure you contribute to it with text or feedback on what other people write.

[x] Submitted to IUI-2015 (by John)

Figure 3.8.: A page from the Organic Data Science Wiki [MGRH15]. Similar to the Darwin Wiki in figure 2.4 the center of the page is containing a Gantt chart providing informations about the current process. Below the Gantt chart the metadata of the current task are displayed, namely Type, Progress, Start date, Target date, Owner, Participants and Expertise. In the left side of the page there is a navigation tree representing decomposed tasks. The tasks progress is indicated with a pie chart enhanced by small red dots which are indicating and overdue task somewhere in the process. In the top navigation bar an icon representing a red bell is informing about overdue tasks which are related to the current user.

## 3.5. Projects in the context of SocioCortex

Understanding SocioCortex as a platform focusing on exposing a REST interface allows the generation of a variety of clients, as well as vertical solutions which are able to use SocioCortex' features and extend them for their own specific use cases. The first project is presenting a client for modeling purposes only. The second approach is about efficient handling and editing the data of SocioCortex, whereas the last project is dedicated to presenting SocioCortex data using MxL and aggregation functions to create visually attractive charts.

### 3.5.1. SocioCortex Modeler

Like IkeWiki described in section 3.3.4 the goal of the SocioCortex Modeler is to enable non-modeling experts creating a model for their business domain specific knowledge [Sch15]. To achieve this the SocioCortex Modeler's focus is to separate the modeling capabilities from other functionalities of the social enterprise application by creating a new client which is dedicated for modeling tasks [Sch15].

The complexity of the client is measured by number of clicks to achieve a certain task. In this case the task was to create the model of a basic pet store consisting of the entities `Address`, `Account`, `Pet`, `Bill` and `Customer`. To get a baseline for the measurements other systems are taken into account which are able to create a modeled environment, too. The competitor with the lowest number of clicks was Tricia and therefore the Hybrid Wiki with 94 clicks [Sch15].

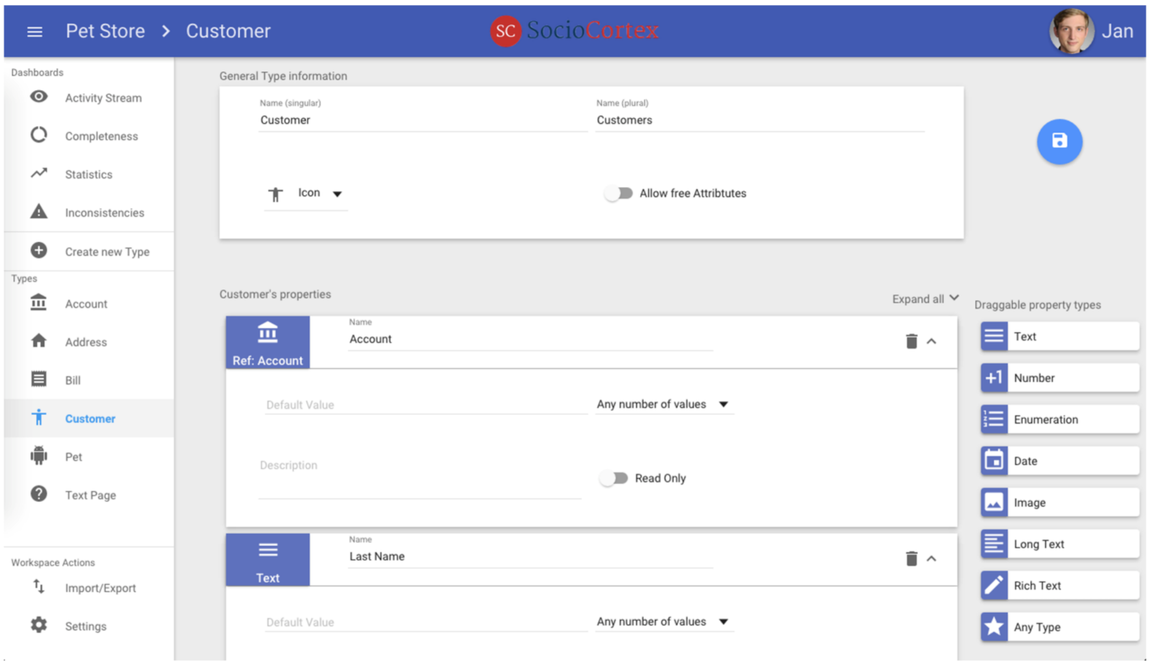
The result of the studies is a client which tries to reduce the functionality until only the needed requirements are supported. This means that only the functionalities for using the type layer are left[2]. A screenshot of the resulting system is shown in figure 3.9.

Using the loose coupling of SocioCortex it is possible to change requirements for types to new or changed needs. Possible value types for attributes besides number, enumerable or date are complex types like `Address` or `Bill`. These complex types are creating references to the assigned informations. Additionally, it is possible to create multiplicities or to set an attribute to read only.

### 3.5.2. SocioCortex Content Manager

The SocioCortex Content Manager addresses business experts who tend to have a less understanding for Information technology (IT) related topics. It is using the phenomenon that every business expert seems to use spreadsheets to persist data [PR00]. Therefore, the solution for the SocioCortex Content Manager is to provide a spreadsheet like environment for managing data of SocioCortex. It is an attempt which is especially targeting complex linked data like they are generated by SocioCortex [Mei15]. This project intends

[2]A short introduction to Tricia's type layer can be found in section 2.1.1



Name (singular): Customer
Name (plural): Customers
Icon
Allow free Attribtutes

Ref: Account — Name: Account
Default Value — Any number of values
Description — Read Only

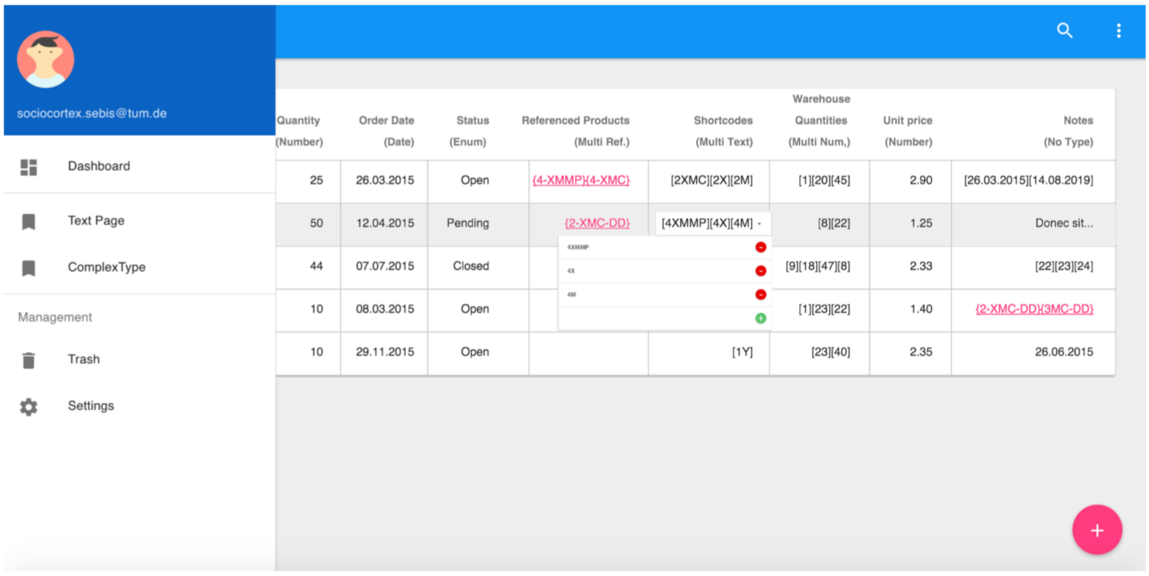
Text — Name: Last Name
Default Value — Any number of values

Figure 3.9.: A screenshot of the SocioCortex Modeler [Sch15]. Unlike the previously shown wiki solutions, this approach does not have any data in its interface. Instead, it is showing the model layer of SocioCortex. The main menu on the left is divided into three parts. General statistics showing the fulfillment of attributes, the types of the current workspace, namely `PetStore`, and at the bottom general settings. The center part of the page is showing the current type `Customer` with its attributes `Account` and `LastName`. New attributes can be added by drag and dropping the designated type to the center of the page. This can be done by using the symbols next to the types listed on the left as well as on the primitive types on the right side of the page. The short text under the attributes symbol having a blue background indicates the type of the attribute. `Account` is of type `Ref:Account` which means the attribute's value have to be a reference to the type `Account`.

for processing structured data only. This means that only the attributes of wiki pages are considered.

The focus of the Content Manager is relying on the nature of spreadsheets tending to have many errors. One possible reason is the lack of automation. Users become strained which increases the error rate [CE08]. Therefore, the Content Manager is using a model-based approach by using the definitions made in the type layer of SocioCortex. For achieving this a developed JavaScript (JS) framework is used for connecting the spreadsheet framework with the type handling of SocioCortex. This combination enables the use of checking constraints while the user is typing in data to keep the right format.

Summing up the features of the SocioCortex Content Manager: The SocioCortex Content Manager provides a spreadsheet like interface within the browser. There is one spreadsheet per type tag (cf. section 2.1.1) representing the instances as rows and type's attributes as columns. The possible values are restricted by constraints which are similar to the constraints made by the SocioCortex Modeler presented in section 3.5.1. Beside simple values the SocioCortex Content Manager is also able to render references as well as to handle derived attributes which means values calculated by MxL statements. An example for the Content Manager can be seen in figure 3.10.



| Quantity (Number) | Order Date (Date) | Status (Enum) | Referenced Products (Multi Ref.) | Shortcodes (Multi Text) | Warehouse Quantities (Multi Num,) | Unit price (Number) | Notes (No Type) |
|---|---|---|---|---|---|---|---|
| 25 | 26.03.2015 | Open | {4-XMMP}{4-XMC} | [2XMC][2X][2M] | [1][20][45] | 2.90 | [26.03.2015][14.08.2019] |
| 50 | 12.04.2015 | Pending | {2-XMC-DD} | [4XMMP][4X][4M] | [8][22] | 1.25 | Donec sit... |
| 44 | 07.07.2015 | Closed | | | [9][18][47][8] | 2.33 | [22][23][24] |
| 10 | 08.03.2015 | Open | | | [1][23][22] | 1.40 | {2-XMC-DD}{3MC-DD} |
| 10 | 29.11.2015 | Open | | [1Y] | [23][40] | 2.35 | 26.06.2015 |

Figure 3.10.: Overall mockup for the SocioCortex Content Manager [Mei15]. A general menu on the left is showing two types: `TextPage` and `ComplexType`. The names of the attributes are shown on top of the spreadsheet. The most left column, currently hidden by the menu, contains the instances' names for distinguishing the different rows. The currently marked cell contains multiple values, which are listed in a little popup window. Each value can be deleted by clicking the corresponding red circle.

### 3.5.3. SocioCortex Visualizer

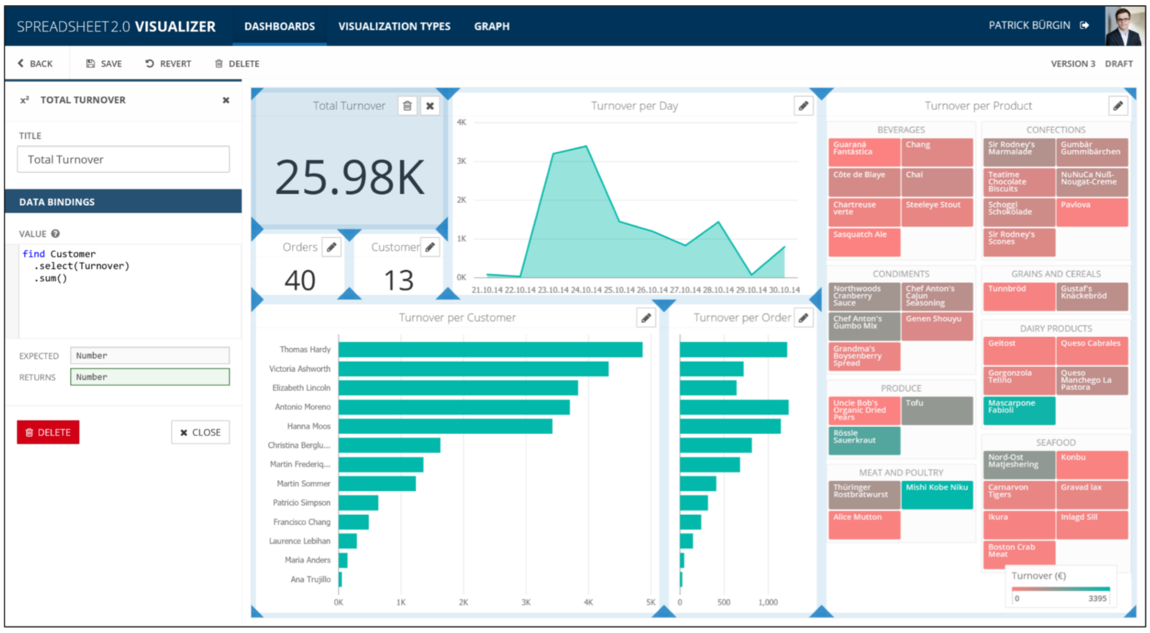
The SocioCortex Visualizer targets aggregation and representation of complex data structures and relations between them. To achieve this, it presents a modular dashboard architecture. The architecture actively supports data flow analysis and facilitates the use of visualization tools [Bür15]. The dashboard itself is mainly consisting of different types of charts which can be customized by the end-user.

In order to create a meaningful dashboard four key requirements are stated [Bür15]:

1. There must be enough flexibility to support different needs across various data domains.
2. The dashboard must be customizable according to the user's needs.
3. A generic schema for specifying different visualization types shall flexible enough to adopt changes of the environment as well as encourage developers to create new tools.
4. Specifications shall include suitable metadata to facilitate identification, discovery and exchange of visualization types.

The resulting dashboard consists of a grid system. Using drag and drop a various number of different types of charts or simply derived values can be shown on the dashboard. An example of a dashboard in edit mode can be seen in figure 3.11

In addition, to the dashboard a traceability environment is provided which allows the user to find relations between the organization's data assets [Bür15]. Basically this environment is drawing a graph depending on the built dashboard. Using the graph facilitates the generation of new charts as well as providing an understanding for the existing graphs. The example of figure 3.11 is continued in figure 3.12 showing the corresponding graph.



Title: Total Turnover

```
find Customer
  .select(Turnover)
  .sum()
```

Expected: Number
Returns: Number

Figure 3.11.: The example dashboard `NorthwindExtended` [Bür15]. The currently marked indicator is backgrounded by a light blue on the upper left corner. It is a simple number derived by the MxL statement shown in the left column. The blue triangles which are in the corners of each chart can be used to resize the charts depending on a grid layout.

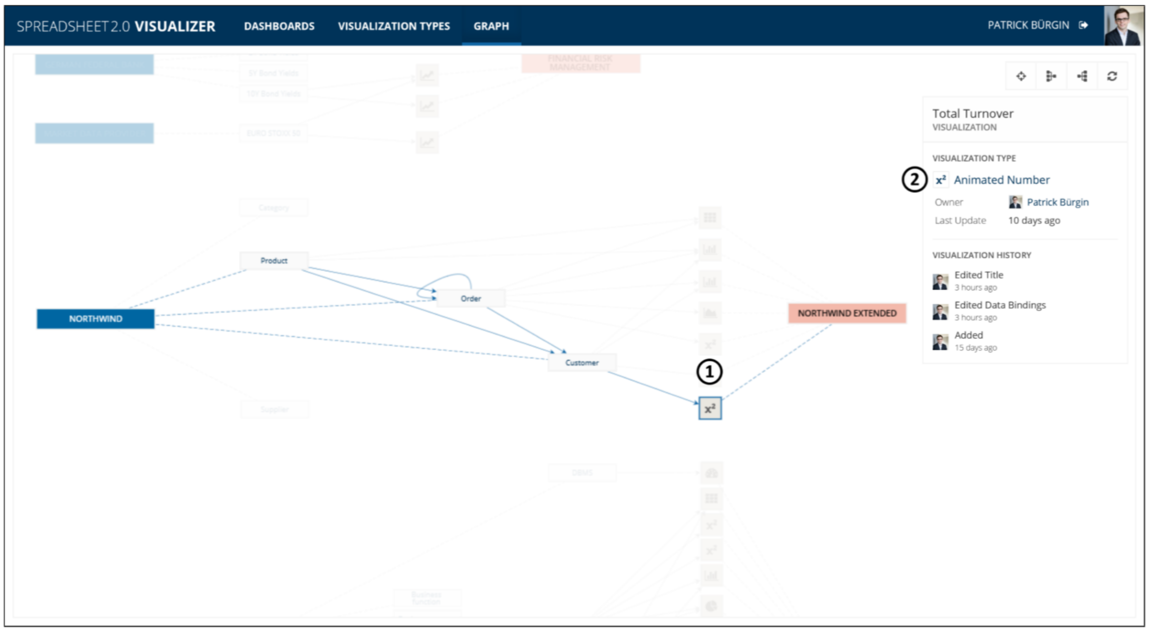


Figure 3.12.: An example for the traceability environment [Bür15]. From right to left the name of the dashboard `NorthwindEXTENDED` is shown. Then the nodes which are representing single charts are followed by types. Types are connected to the workspace `Northwind`. In this particular image the user clicked on chart symbol at (1) which highlights all paths related to this node. In addition, the visualization type is shown in the window on the right side at (2). In this way the user can click on different nodes in order to see the relations and navigate through them.

# 4. Used Technologies and Libraries

The following chapter illustrates used technologies and libraries. First the setup for the SocioCortex Generic Client is introduced along with its dependencies. After this, the SocioCortex persistence layer is introduced and described to get a deeper understanding for further implementations. In addition, the REST Application Programming Interface (API) is described to provide a background knowledge for the adoptions made in chapter 6.

## 4.1. Front end Technologies

Due to the strict separation of the clients and SocioCortex which is above all providing a REST API it is possible to create two strictly distinguished projects. This section presents technologies and libraries used for development as well as for the client itself.

### 4.1.1. NodeJS Package Manager

The NodeJS Package Manager (NPM) is a package manager shipped with NodeJS[1]. A package manager is like a third-party package repository, second it helps to keep an overview over the installed packages and third it enables to define dependencies on other packages and to resolve them as needed [Tei13]. As a resource for packages in order to find new packages or useful dependencies the website[2] can be used as it is using the same centralized repository as NPM itself.

The SocioCortex Generic Client uses NPM to manage its development environment. Packages used are noted in the `package.js` file. Using NPM the following packages are included:

**gulp**[3,4] for build management and processing using a stream like system (cf. section 4.1.2).

**gulp-filter**[5] to create filtered subsets that can be used in gulp's streams.

**gulp-inject**[6] enables to create placeholder in the targeting build file to inject further dependencies.

[1]see `http://nodejs.org/`

[2]see `http://www.npmjs.com/`

[3]see `http://www.npmjs.com/package/gulp`, accessed on January 06th 2015

[4]see `http://github.com/gulpjs/gulp/`, accessed on January 06th 2015

[5]see `http://www.npmjs.com/package/gulp-filter`, accessed on January 06th 2015

[6]see `http://www.npmjs.com/package/gulp-inject`, accessed on January 06th 2015

**gulp-order**[7] to reorder the files in a stream

**gulp-sass**[8] provides a preprocessor for SASS and SCSS files and returns the CSS to gulp's stream.

**gulp-webserver**[9] runs a local web server including a live reload feature enabling continuous development without needing to reload the web page manually.

**gulp-watch**[10] is continually watching a files set and runs an action if a file changes. In the context of the SocioCortex Generic Client this means that every time a file like an SCSS file is changed gulp-watch will initialize a rebuild. gulp-webserver recognizes the changed built files and reloads the web page.

**del**[11] used to clean up temporary files created during the build.

**bower**[12,13] enables using bower as web technologies dependency management.

**main-bower-files**[14] used for getting bower's dependencies and include them to the build process.

### 4.1.2. gulp

In the SocioCortex Generic Client gulp is used to manage the build process of the application. Gulp is a stream-based building tool which's configuration files are written using pure JavaScript [Dic15]. However, gulp is focusing on streams, every stream is handled within a task and every task can depend on another task. An example for a task can be seen in listing 4.1. The responsible file for the building process is called by default `gulpfile.js`.

During development normally the task `webserver` is started which starts a web server including a watch feature. The watch feature observes the whole source directory and recognizes file changes. If a file change occurs the building process is re-launched.

Even though gulp is running with JavaScript and therefore uses event-based asynchronous processes a certain order can be determined due to the dependencies of the single tasks. The building process of the SocioCortex Generic Client includes 5 different steps:

**clean**

The `clean` task deletes the temporary folder `./tmp/` which is containing all data generated of previous executings.

[7] see `http://www.npmjs.com/package/gulp-order`, accessed on January 06th 2015
[8] see `http://www.npmjs.com/package/gulp-sass`, accessed on January 06th 2015
[9] see `http://www.npmjs.com/package/gulp-webserver`, accessed on January 06th 2015
[10] see `http://www.npmjs.com/package/gulp-watch`, accessed on January 06th 2015
[11] see `http://www.npmjs.com/package/del`, accessed on January 06th 2015
[12] see `http://bower.io/`, accessed on January 06th 2015
[13] see `http://www.npmjs.com/package/bower`, accessed on January 06th 2015
[14] see `http://www.npmjs.com/package/main-bower-files`, accessed on January 06th 2015

```
1 gulp.task('sass', ['clean'], function () {
2 gulp.src(['./src/css/*.css','./src/css/*.scss'],{base:'./src/css'})
3     .pipe(sass().on('error', sass.logError))
4     .pipe(gulp.dest('./tmp/css'));
5 });
```

Listing 4.1: A typical gulp task mentioned in `gulpfile.js`. The task `sass` is depending on `clean` which makes gulp to run the `clean` task before the `sass` task. The stream of the task is started in line 2 by searching all files matching the pattern. The resulting files are piped to the SASS preprocessor which is rendering the files to valid CSS. The last step in line 4 sends the files to a temporary destination for further processing.

**bower**
: This task searches for resources like JavaScript, CSS or image files which are managed by bower dependencies and copies them to the temporal destination `./tmp/` folder.

**sass**
: The `sass` task is the task explained in listing 4.1. It searches for CSS and SCSS files in the folder `./src/css/` and sends them to the same temporal destination folder like the bower task.

**inject**
: The next step includes injecting the temporal files into an `index.html` file. The injection is made using relative paths to the JavaScript files. Another possibility would be to inject the file contents directly to avoid multiple calls to the server. Including all possible files into a single HTML file is useful when being in the productive phase.

**build**
: The final build tasks launches the previous tasks and takes left images from `./src/images/` and moves them to the dedicated folder `./tmp/images`, too.

### 4.1.3. Bower

Bower is a package management system similar to NPM explained in section 4.1.1. The focus of bower is on web dependencies like JavaScript libraries or CSS frameworks. The SocioCortex Generic Client uses bower for injecting the core AngularJS library and extension of AngularJS explained in section 4.1.4.

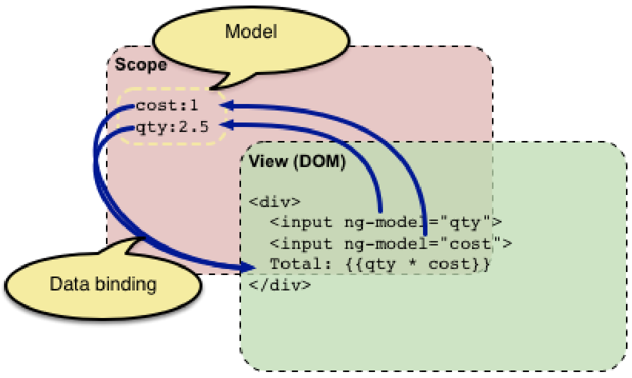
### 4.1.4. AngularJS

AngularJS[15] is a JavaScript framework powered by Google. The basic idea of Angular is to use the Model View Controller (MVC) pattern in a different way than classical web pages do. Controller and view layer are handled by the client which enables immediate responses for the user. Only the model and persistence layer is provided on the server. Therefore, Angular treats HTML pages as templates which can be adjusted on runtime. The concept of AngularJS relies on three points [Ang15a]:

**Two-way data binding**
enables to change informations in the JavaScript held model, which are then reflected in the HTML template. Therefore, new markup is introduced shown in figure 4.1.

**Controllers**
are used to handle more complex business logic. The purpose of controllers is to expose variables and functionality to expressions and directives.

**Services**
are the connection to a persistent back end. They are responsible of sending and receiving data from any distributed source.



```
<div>
  <input ng-model="qty">
  <input ng-model="cost">
  Total: {{qty * cost}}
</div>
```

Figure 4.1.: An example that shows the two-way data binding using AngularJS specific markup [Ang15a]. The regular HTML input fields are annotated with a new attribute called ng-model. This annotation creates variables in the scope of the application. The third row represents another part of Angular's markup. Using two curly brackets the scope's model can be accessed. In this example the two numbers are multiplied. Changing the values of the inputs (green) results in a change of the model's data (red). This launches an update of the multiplying expression, showing the new result.

[15]see `https://angularjs.org/`, accessed on January 06th 2016

The SociocCortex Generic Client uses besides AngularJS itself some extra modules of the framework:

**ngMessages**[16]
is used for generation of messages which are shown from a key/value collection. This extension is mainly used for showing error message if the validation of inputs fails. An example can be seen in listing 4.2

**ngSanitize**[17]
sanitize HTML markup received from insecure sources which enables to insert content persisted on the server like wiki page. Without using ngSanitize it is not possible to include foreign HTML

**ngRoute**[18]
enables so called deeplinking. Deeplinking is a method for using the browser's hash as a way to navigate through the application without causing reloads of the application.

```
<form name="myForm">
  <label>
    Enter text:
    <input type="text" ng-model="field" name="myField" required
        minlength="5" />
  </label>
  <div ng-messages="myForm.myField.$error" role="alert">
    <div ng-message="required">You did not enter a field</div>
    <div ng-message="minlength, maxlength">
      Your email must be between 5 and 100 characters long
    </div>
  </div>
</form>
```

Listing 4.2: An example for using the ngMessage package. As long as the user does not provide an input the `$error` variable stays empty. As soon as the input field's value changes the validation constraints `required` and `minlength` are evaluated. If either of these constraints is violated the corresponding message is shown beneath the input field.

[16] see `http://docs.angularjs.org/api/ngMessages`, accessed on January 07th 2016
[17] see `http://docs.angularjs.org/api/ngSanitize`, accessed on January 07th 2016
[18] see `http://docs.angularjs.org/api/ngRoute`, accessed on January 07th 2016

### 4.1.5. Angular Material

Material Design is a design language created by Google. Its goal is to provide a universal design guideline across platforms by keeping classic principles of design synthesized with today's technologies [Goo15a]. Its name is used as a metaphor because every layer of an application shall have the look of a paper laying over another. It states that the key for interacting with objects is visual look and feel depending on light, surface and movement.

Beside the name-giving material effects, Material Design aims at keeping things simple and understandable, which results in very big buttons often only titled by a clean and bold symbol. The third goal is to introduce the user as the "prime mover" [Goo15a]. This means, that the whole interface of the application shall fit on one single environment to provide a meaningful and appropriate user experience [Goo15a].

Angular Material[19] is the reference implementation of Google's Material Design Specification. It combines the look of Material Design with modules for building web applications using AngularJS. For this purpose, several User Interface (UI) components are introduced. The majority of these components are either structuring components like tabs, menus or cards or they are input elements like date-pickers, customized radio buttons or switches.

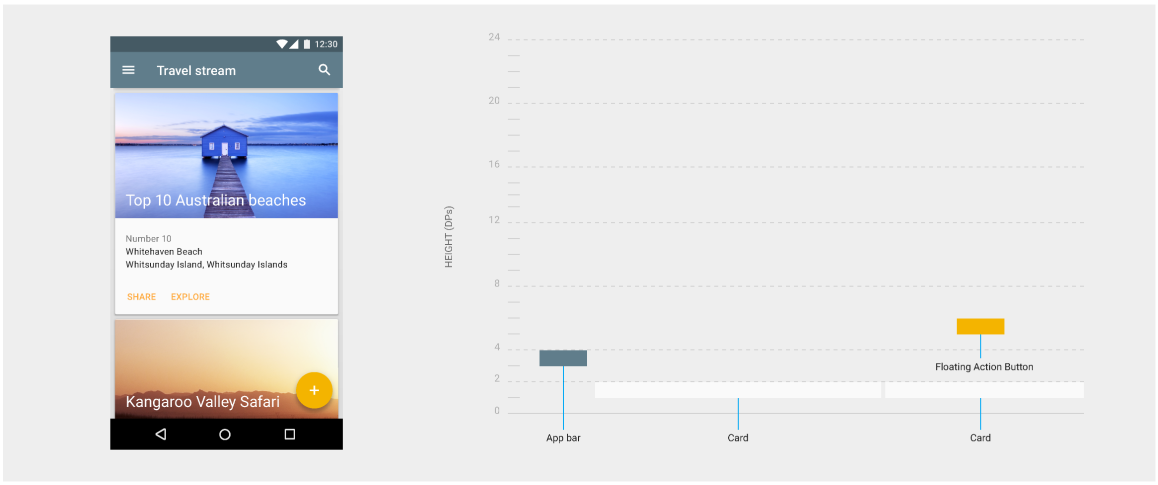


Figure 4.2.: An conceptual approach of Material Design explaining the 3D effects and different layer of an application [Goo15b].

### 4.1.6. scAngular

The library scAngular is an AngularJS module, which's enables using predefined services. The goal is to implement all REST API interfaces for providing a bridge between Socio-

[19]see `http://material.angularjs.org/`, accessed on January 07th 2016

Cortex' back end and the front end. Therefore, front end developers do not have to care about this connection. The following shows the implemented functionalities as of the current implementation status[20] which is a module containing 6 different services according different purposes. Every of them is implementing a various number of resources, which some of these are built using `ngResource` which enables to use all possibilities `ngResource` offers[21].

The services of `scModel`, `scData` and `scPrinciple` are using `ngResource`. The default resources made by ngResource do not have a method like `update`. Due the fact that almost every resource of these services implements the `update` method it is not explained again in the following part. `Update` is using a `PUT` request to alter objects. The method `query`, which is implemented by default by ngResource is often deleted. A third often occurring customization is the object attribute `id` being used as identifier. These three customization will not be mentioned in the further description unless there are deviations. The chosen Hypertext Transfer Protocol (HTTP) method for custom methods is shown within square brackets. For example, `[GET]queryByWorkspace` indicates that `queryByWorkspace` uses the `GET` method, which also indicates that it is returning a list of objects according to the documentation of ngResource [Ang15b].

### scAuth

This service implements methods for authentication handling using SocioCortex. Purposes like login, logout or getting user details are taken care within this service. scAuth is the only service which does not use the ngResource module.

**`login(user,password,callback,error)`**
: returns a promise. It is successful if the given user credentials are valid. If `call back` and `error` are valid functions they can be either used as normal callbacks or as transformation functions for the result of the promise.

**`logout()`**
: returns `undefined`. It deletes credentials and details from the local storage but does not remove them from the default http headers. This means that this method must be followed by `setAuthorizationHeader()` in order to disable the client form interacting with SocioCortex.

**`getUser()`**
: returns the user from the local storage, which is set by the `login` method. If no user is set, `getUser()` will return `undefined`.

**`isAuthenticated()`**
: returns a boolean whether the user details in the local storage are existing or not.

[20] last commit was #b536e8a on January 13th 2016

[21] see the documentation of ngResource at `http://docs.angularjs.org/api/ngMessages`, accessed on January 07th 2016

**`setAuthorizationHeader()`**
returns undefined. If the local storage is containing credentials, which were set using `login()`, this method sets the default http authentication header in order to be sent with every request. The method `login()` will call this method automatically.

**scModel**

scModel is a service for providing access to the type layer of SocioCortex. There are four resources supported:

**EntityType**
adds the methods

**`[GET]queryByWorkspace`** retrieves a list of `EntityType`s by calling `/workspaces/:id/entityTypes` using a workspace id.

**`[GET]getAttributeDefinitions`** retrieves a list of `AttributeDefinitions`s, which are assigned to the specified `EntityType`

**`[GET]getEntities`** retrieves a list of objects of type `Entity`, which are assigned to the specified `EntityType`

**AttributeDefinition**
This resource implements the following method:

**`[GET]queryByEntityType`** which retrieves a list of objects of type `AttributeDefinition` which are assigned to a specified `EntityType` by an given `id`

**DerivedAttributeDefinition**
`DerivedAttributes` are attributes with values are calculated using the values of normal attributes. This resource helps to manage these and further offers the following method:

**`[GET]queryByEntityType`** which retrieves a list of objects of type `DerivedAttributeDefinition` which are assigned to a specified `EntityType` by an given `id`

**CustomFunction**
Beside the standard methods this service includes the following:

**`[GET]queryByEntityType`** works like `queryByEntityType` of the resources above except it is retrieving a list containing objects of type `CustomFunctions`

### scData

The service of scData is responsible for managing the instance layer of SocioCortex. The supported resources are:

**Entity**
: The entity `Entity` is representing the wiki pages of SocioCortex. In addition to the methods of ngResources the following methods are added:

**`[GET]queryByEntityType`** retrieves a list containing objects of type `Entity` given a `EntityType` id.

**`[GET]queryByWorkspace`** retrieves a list containing objects of type `Entity` of a specified `Workspace` id.

**`[GET]getAttributes`** retrieves a list containing objects of type `Attribute` using the id of the `Entity`.

**`[GET]getFiles`** is a method to access metadata of files attached to an `Entity`, which can then be used to download or upload a specific file.

**File**
: The `File` resource manages files and file metadata.

**`[GET]queryByEntity`** retrieves a list of objects containing all metadata of files attached to the given `Entity` id.

**`[GET]download`** initializes the download of a file itself using a `File` id.

**Workspace**
: This resource manages workspaces and helps to retrieve directly related data like `Entity`s and `EntityType`s.

**`[GET]getEntities`** uses the `Workspace` id to request all its `Entity` objects.

**`[GET]getEntityTypes`** like `getEntities` but expects `EntityType` objects form the server.

**Attribute**
: This resource manages `Attribute`s which are always attached to an `Entity`. Addition methods are:

**`[GET]queryByEntity`** returns a list of objects of type `Attribute` which are assigned to the given `Entity` id.

**Task**
: The task resource connects the task functionalities described in this thesis to the SocioCortex Generic Client. Beside for standard managing purposes this resource offers method like the following:

**`[GET]getAttributes`** requests `Attribute`s similar to `Entity.getAttributes()` except these one are in addition attached to an Task

**Expertise**
This resource only uses the standard managing methods provided by ngResource under the constraints mentioned above.

### scMxl

The `scMxl` service is implemented for use in the SocioCortex Visualizer as well as for the SocioCortex Modeler. Therefore, the methods of this service are not following the ngResource guidelines similar to the `scAuth` service. All of these methods have `callback` and `error` parameter, which are used similar to `scAuth.login()`.

**`autocomplete(context,restriction,callback,error)`**
provides informations to be used for autocompletion.

**`query(context,data,callback,error)`**
runs a MxL query stated in the `context` on the server and retrieves the result.

**`validate(context,data,callback,error)`**
validates a MxL statement on the server and returns the response of the server.

### scPrincipal

The service `scPrincipal` is used to manage user related data. Unlike `scData` and `scModel` the resources specified here do implement the `query` method.

**User**
This resource provides methods for managing user related data. In addition it provides the following methods:

**`[GET]picture`** returns the picture of a user specified with the user's id.

**`[GET]me`** returns informations about the currently logged in user, using the credentials provided by the `scAuth` service.

**`[GET]myPicture`** returns the profile image of the currently logged in user, again using the credentials provided by the `scAuth` service.

## 4.1.7. Building the development environment

The project of the SocioCortex Generic Client is hosted on Bitbucket.org. Therefore, to start with developing the SocioCortex Generic Client it is necessary to check out the client[22]. After checking out the client NPM is needed to manage further development dependencies. After NPM is installed running `npm install` within the folder of the checked out project

[22]see `https://bitbucket.org/sebischair/sc-client-generic`, accessed on January 14th 2016

all further dependencies should be installed automatically. Running `npm install -g` in the command line would install the dependencies globally on the system.

After NPM has installed all dependencies it should be possible to run `bower install` within the command line which will scan for `bower.json` and install all listed dependencies. The final step is to call `gulp` which will start the web server showing the project.

## 4.2. Persistence Layer of SocioCortex

The persistence layer of SocioCortex is a custom Object-Relational Mapping (ORM) implementation. An complete overview for the ORM layer is provided in figure 4.3. It is automatically adding and altering database tables according to their Java class representation. Therefore, on every start of the SocioCortex platform the implementation of the persistence layer is checking for classes which are added to `PlatformPlugin`. The class `PlatformPlugin` itself holds an array of all classes which are representing entities. Therefore, the first requirement for newly added entities is to register within the array stated in `PlatformPlugin`. Furthermore, a class which is representing an entity has to fulfill two more requirements in order to function properly.

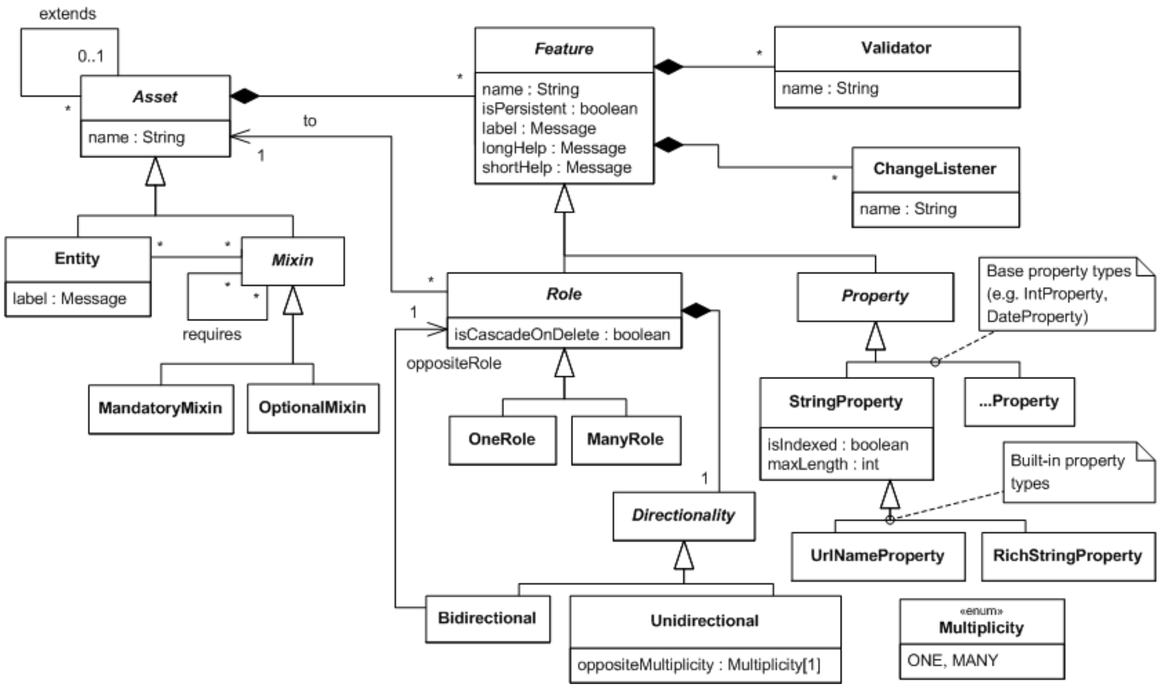


Figure 4.3.: The ORM layer model of SocioCortex [Neu12].

Second requirement for a class representing a entity is to inherit from `PersistentEntity` which is then providing basic CRUD (Create Read Update Delete) functionality. The attributes of the entity are fetched using reflection methods and automatically added to the entities in the database.

An attribute itself is represented by an class inheriting one of the subclasses from `Property`. Since the subclasses are all abstract classes it is necessary to create an extra class for each attribute. For saving boilerplate code Java's concept of anonymous classes is used instead of creating new classes. An example concerning attributes and their implementation is illustrated in listing 4.3. These anonymous classes are able to implement further constraints which are also detected by reflection methods.

The third and last requirement is to provide a `SCHEMA`. An example schema is presented in listing 4.3 in lines 7 and 8. With `PersistentSchema` again being an abstract class it enforces to create a new anonymous class in order to provide an instance.

### 4.2.1. Implementing Associations in SocioCortex

Associations are implemented using instances of the abstract class `Role`. There are two different implementations: `OneRole` and `ManyRole`. In listing 4.3 `Stage` contains an instance of `ManyRole`. It is pointing to its opposite role which is implemented by providing an instance of `OneRole`. SocioCortex is interpreting this code as an one-to-many association. Consequently, the database will be altered by adding a new attribute to the entity containing the `OneRole` instance. Regarding this example the entity of `TaskDefinition` will have added a column called `stage`. If the opposite role is not intended to implement an association, it is also possible to simple add the multiplicity of the counterpart by overriding the method `oppositeMultiplicity` instead of `oppositeRole`.

Another feature of the role concept is the ability of cascading deletes. If the deletion of an entity shall initialize the deletion of an associated entity, the first one has to override the method `isCascadeDelete` with returning the value `true`. With regard to listing 4.3 this means if the deletion of a `Stage` should cause removing of all associated `TaskDefinitions`, `Stage` must implement the method.

The third feature of `Role`s is the possibility of adding validation methods to entities. These are added with adding methods to the anonymous classes which are returning an instance of type codede.infoasset.platform.orm.Validator. The name of these methods is not important because they do not override any method. Instead, again Java's reflection capabilities are used to find all methods returning an instance of `Validator`.

## 4.3. REST API Layer of SocioCortex

In SocioCortex the response of a HTTP request is created by a station. Stations are providing different formatted outputs such as a redirection to other resources or the delivering of data. For the latter purpose an own super class is introduced because these deliverable

```
 1 package de.infoasset.platform.assets.task;
 2 
 3 import de.infoasset.platform.orm.*;
 4 import de.infoasset.platform.rest.RestApiPatterns;
 5 
 6 public class Stage extends PersistentEntity {
 7   public static final PersistentSchema<Stage> SCHEMA =
 8     new PersistentSchema<Stage>() {};
 9 
10   public final ManyRole<TaskDefinition> taskDefinitions =
11     new ManyRole<TaskDefinition>() {
12       protected OneRole<Stage> oppositeRole() {
13         return TaskDefinition.SCHEMA.prototype().stage;
14       };
15     };
16 
17   public String getResourceType() {
18     return RestApiPatterns.STAGES;
19   };
20 
21   public final StringProperty name = new NameProperty() {};
22 
23   @Override
24   public String getName() {
25     return name.get();
26   }
27 }
```

Listing 4.3: The complete entity `Stage`. The inheritence of `Process` was replaced by the general super class `PersistentEntity` to provide a better understing for the implementation. The attribute `name` would normally be implemented inside of `Process`. `taskDefinition` and `name` are examples for anonymous classes. In `taskDefinition` the method `oppositeRole` is implemented. The method is pointing to the `Role` which is implementing the counterpart of an association. Lines 17 to 19 are providing a method for use in the REST API layer explained in section 4.3.

data are further differentiated. The class is called `AnswerStation` and is producing all possible outputs for the system. In the Hybrid Wiki the class `AbstractPage`, which is a subclass of `AnswerStation`, provides templates which are rendered and transported to the user.

For SocioCortex and its REST API a new `AnswerStation` was implemented for providing services in a RESTful context. The basic techniques for creating a new component in the API are explained in the following.

`RestApiPatterns` is the class where all routes are connected to the dedicated handling classes. This is done by searching for `HandlerPattern`s which are attributes holding a route, consisting of a string, and the associated `RestApiHandler`. The `HandlerPattern`s are found using reflection techniques. In order to create a new resource in the RESTful context it is necessary to register the desired route in this class.

The `RestApiHandler` is providing HTTP methods like `GET`, `PUT`, `POST` and `DELETE`. Without overriding these methods the `RestApiHandler` will answer using the code 501 which is the HTTP error code for *not implemented*. Therefore, by overriding one of the methods the developer is able to deliver JavaScript Object Notation (JSON) serialized objects. The instances of the `RestApiHandler` are also responsible for fetching the right objects from the database as well as for persisting objects which are sent by the client. Since the exchange format is JSON there are further classes for managing the serialization.

Every provided entity of the API implements its own serializing class. All instances of `RestApiSerializer` are registered in this super class. This enables the possibility of generic serializing and deserializing.

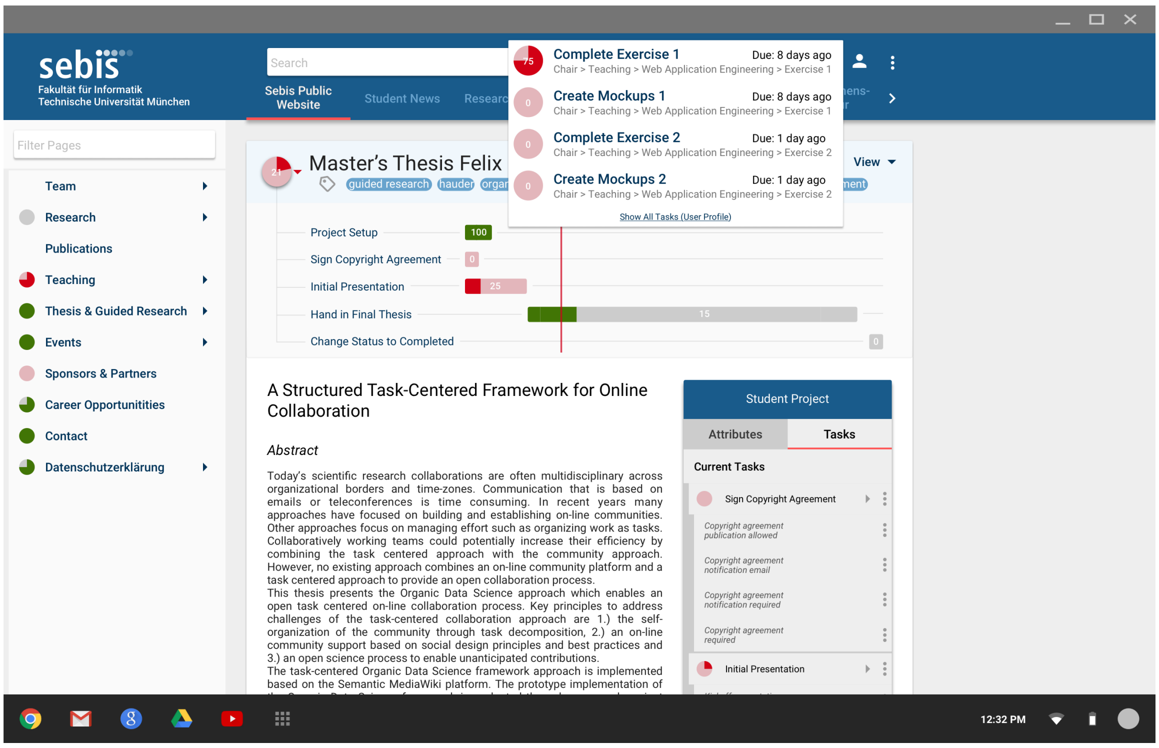
# Part II.

# Contribution

# 5. Generic Client for SocioCortex

SocioCortex as well as the Darwin Wiki are based on the Hybrid Wiki. The difference is, that SocioCortex' approach is not to provide a user interface where all use-cases are integrated than just being an API enabling programmers to easily create their own user interfaces.

The implementation of the SocioCortex Generic Client is an interface providing the basic features shown in chapter 2 and chapter 6. This chapter shall introduce the added functionality as well as the use cases which are going to be supported.



| | | |
|---|---|---|
| 75 | Complete Exercise 1 | Due: 8 days ago |
| | Chair > Teaching > Web Application Engineering > Exercise 1 | |
| 0 | Create Mockups 1 | Due: 8 days ago |
| | Chair > Teaching > Web Application Engineering > Exercise 1 | |
| 0 | Complete Exercise 2 | Due: 1 day ago |
| | Chair > Teaching > Web Application Engineering > Exercise 2 | |
| 0 | Create Mockups 2 | Due: 1 day ago |
| | Chair > Teaching > Web Application Engineering > Exercise 2 | |

Show All Tasks (User Profile)

A Structured Task-Centered Framework for Online Collaboration

*Abstract*

Today's scientific research collaborations are often multidisciplinary across organizational borders and time-zones. Communication that is based on emails or teleconferences is time consuming. In recent years many approaches have focused on building and establishing on-line communities. Other approaches focus on managing effort such as organizing work as tasks. Collaboratively working teams could potentially increase their efficiency by combining the task centered approach with the community approach. However, no existing approach combines an on-line community platform and a task centered approach to provide an open collaboration process.

This thesis presents the Organic Data Science approach which enables an open task centered on-line collaboration process. Key principles to address challenges of the task-centered collaboration approach are 1.) the self-organization of the community through task decomposition, 2.) an on-line community support based on social design principles and best practices and 3.) an open science process to enable unanticipated contributions.

The task-centered Organic Data Science framework approach is implemented based on the Semantic MediaWiki platform. The prototype implementation of

Current Tasks

- Sign Copyright Agreement
  - *Copyright agreement publication allowed*
  - *Copyright agreement notification email*
  - *Copyright agreement notification required*
  - *Copyright agreement required*
- Initial Presentation

Figure 5.1.: Mockup for the SocioCortex Generic Web Client as part of a bachelor's thesis [Kat15].

## 5.1. Use Cases

Mockups as in figure 5.1 are created by Florian Katenbrink as contribution of his bachelor's thesis. They are supposed to be a template for the later implementation and are hereby used to help defining the use cases.

The design is based on the design of the two previous solutions. The Hybrid Wiki as well as the Darwin Wiki have a navigation tree on the left side of a page providing the ability for navigating through the contents of the page. Attributes on the right side are representing the structured content of a page, whereas the unstructured content is placed in the middle.

The following sections are describing in detail the use cases which are implemented by this thesis.

### 5.1.1. Task Navigation

As well as Tricia and Darwin the web client shall have a navigation tree on the left side but instead having a pure navigation tree, this one is extended by pie charts every representing the current progress of the page which is calculated similar to Darwin's page progress calculation described in section 2.2.2. Using the navigation tree, the end-user shall be able to drill down pages to find pages which need further contribution like fulfilling of attributes.

### 5.1.2. Task Planning

Tasks in the SocioCortex context are having multiple metadata to describe the task itself like it is in the Darwin Wiki, too. The main difference of the metadata in Darwin to the metadata in SocioCortex is that there is no `delegatedto` attribute. Instead in SocioCortex the end-user have the possibility to define an `owner` attribute of an task. The owner is responsible for the task but does not have the duty to complete the task himself.

Complete fulfillment of the metadata enables the end-user to plan the sequential order of the tasks by using the Gantt chart. This chart enables the user to see at first glance which tasks need to be done or are already done. This enables him to check if the schedule still fits reality. For correctly displaying the task in the Gantt chart at least the begin and end date have to be defined. Inconsistence of a task is also possible, which means that the task begins or ends out of the begin and end date range defined by the parent task.

### 5.1.3. Task Execution

The client supports the end-user executing a task with completing the deliverables which can be done by either writing the results to the page's unstructured content section in the middle of the page or filling out the attributes of the box on the right side of the page. For both use cases the client offers a solution.

The text section of the wiki page contains plain HTML markup. Assuming that end-users in general do not know how to create HTML markup a HTML WYSIWYG editor is used in order to support end-users to create content and to prevent a deficit in motivation for adding content because of a lack in programming knowledge.

The attributes section on the right side containing the structured data of the page provides editing features like in Darwin or Tricia. The differences are the visualization of page attributes and task attributes. Two tabs are introduced, the left tab is listing all attributes of a page similar to Tricia. Whereas the second tab which is show in figure 5.1 contains primarily a list of all tasks of the page.

The first line of every task block contains a pie chart indicating the task's progress. Afterwards the task's name is shown followed by two clickable icons. The first icon is a little arrow. Clicking on it extends a new area directly beneath the line of the task's name. This area shows the task's metadata like explained in section 2.2.2. The last part of a task block contains a list of the task's attributes. The mechanics and the look and feel of these are the same as the list for the page's attributes. After the attributes the next task is shown.

### 5.1.4. Task Finishing

A process is "a series of actions or steps taken in order to achieve a particular end" [Oxf16b]. In the context of SocioCortex the "particular end" of a process often will be to provide the requested deliverables. Therby, deliverables are attributes of a task or contribution to the wiki's page. However, in the context of KIPs it may occur that some deliverables are never created or even not intended to be created. Therefore, two different possibilities exist for further progress.

First setting the progress of a task to 100% will always cause it to be finished. If a user sets a task's progress to another value than 100% SocioCortex will not automatically recognize the finishing of this task anymore. The mechanism for updating the progress providing deliverables is deactivated by changing the value by hand.

The second possibility is that the user decides that a task does not have to be finished. Aborting a task is not possible but the user is able to skip it. Skip thereby intends that the current task is not going to be finished. The reason could be that the task was obsolete or the conditions of the execution environment changed such as new terms of rights were introduced.

### 5.1.5. Process Adaption

SocioCortex is continuing the loose coupling of templates to instances because of the same needs of case and exception handling like in the Darwin Wiki [HKM15, SM95]. Consequently, the client allows methods to adapt a page's process to the current needs of the user.

If the predefined tasks are not sufficient for the process there is the possibility of adding a new task to the page. The created task will not have any task definition attached to it and therefore have no attributes, too. Nevertheless, it is possible to assign attributes by entering the attribute title in the field below the task's attributes. An auto-completing list will appear and by choosing the desired attribute the user assigns this attribute to the task.

Existent attributes sometimes are not describing the process close enough. The user is then able to add new attributes. These attributes can be assigned to task as any other attribute to.

In other situations some attributes may be cumbersome and obsolete. For this purpose the user is able to remove an attribute from a task which means it is still available as a page attribute but it can be deleted from the page's attributes list as well.

### 5.1.6. Task Notifications

Sometimes it may happen that a user forgets to finish a task. For this purpose a notification icon is introduced in the top right corner of the UI. The number of overdue tasks will be displayed within a little red circle overlaying the bell icon like in figure 5.1.

Clicking on the bell icon opens a list with clickable items. Every of the items will be linked to the correspondent page of the overdue task.

## 5.2. Structure of the SocioCortex Generic Client

The Generic Client is built with the use of AngularJS which facilitates the separation of concerns, similar to the structure of packages in Java.

### 5.2.1. Architecture of the SocioCortex Generic Client

Using the concept of directives provided from AngularJS it is possible to split the application into disjunct parts which can be easily included into different parts of the application. The single components are obtained using the mockup of figure 5.1.

At first glance the application presented in the mockup is able to be divided into three main sections. A navigation bar is placed on the top, providing navigation between workspaces. A secondary navigation for navigating within a workspace consisting of a hierarchical list is placed on the left side. The third part is the center part in the middle of the page. However the first two sections are implemented, this thesis on the center part of the showing wiki page and will be explained in the following section.

### 5.2.2. Managing Task Representation

Representing tasks is one objective of this thesis. Therefore, different components are created to support this. The first component is `Attributes`, which is representing a wiki page's attributes in the right side of the mockup presented in figure 5.2.

The component is realized by introducing a single directive called `scAttributes` which is used to present both, the page's attributes as well as the task's attributes in order to provide the tab view. The serialized entity using JSON is passed to the directive's attribute `entity` in order to get access to both, tasks and attributes. Another attribute of the directive is called `onChange` which is a method to be called when something of the entity is changed. This includes every change possible by the `scAttributes` directive like changing attributes and their values as well as metadata of tasks.

A module made for representing tasks is the module `scCharts`. This module provides directives for displaying charts in order to enhance the comprehension of the end-user for the view of the page. It is providing two directives called `scGanttChart` and `scPieChart`. The first one is used for rendering the Gantt chart shown in figure 5.2. To achieve this an array of serialized `Tasks` is expected.

The second directive is called `scPieChart`. It is representing a simple pie chart for displaying a value between 0 and 1. It has various attributes for customizing the rendered pie chart:

**`percentage`**
: is representing the number to show and must be between 0 and 1.

**`foreground`**
: to provide a foreground color.

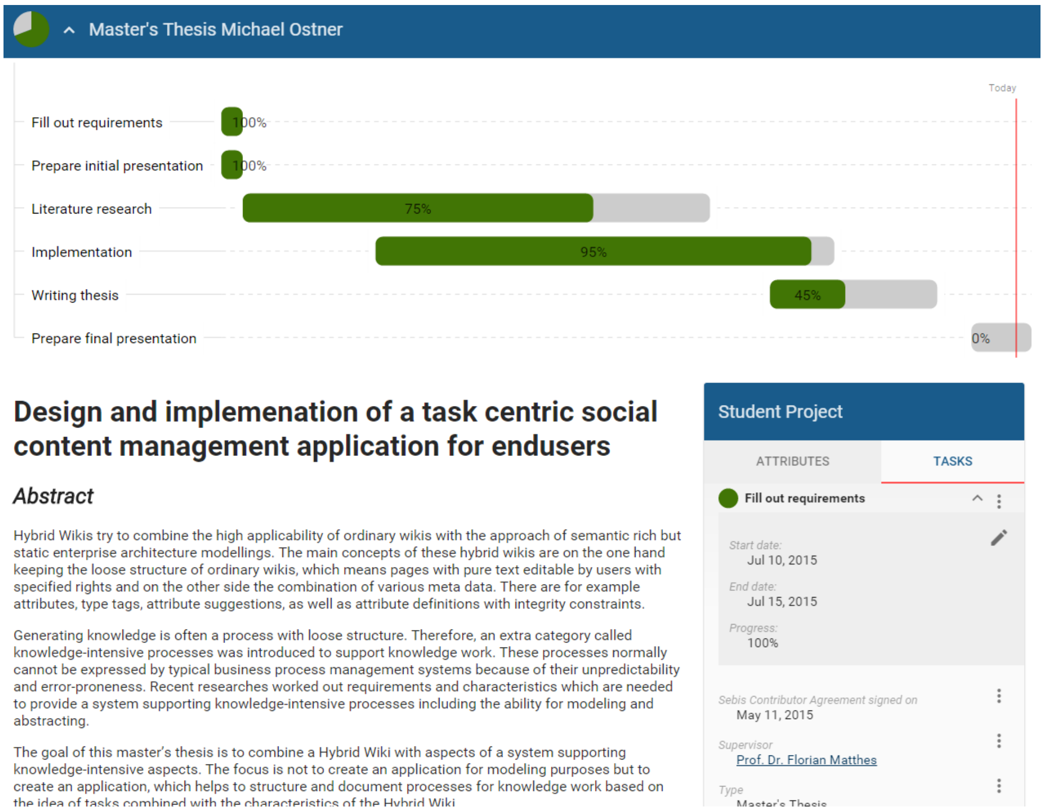
**`background`**
: for a background color.

**`diameter`**
: to define the size of the chart. By default this value is set to `0px`

**`showPercentage`**
: to show the presenting percentage by a number in the center of the chart. `true` by default.

Beside the attributes `foreground` and `background` of `scPieChart` another possibility to style both, `scPieChart` and `scGanttChart` are the provided CSS classes defined in `/src/css/scCharts.scss`.

The module `scCharts` is using plain Scalable Vector Graphics (SVG) and AngularJS techniques like two-way data binding. This enables the chart to be reactive according to changes in the data model.



Design and implemenation of a task centric social content management application for endusers

*Abstract*

Hybrid Wikis try to combine the high applicability of ordinary wikis with the approach of semantic rich but static enterprise architecture modellings. The main concepts of these hybrid wikis are on the one hand keeping the loose structure of ordinary wikis, which means pages with pure text editable by users with specified rights and on the other side the combination of various meta data. There are for example attributes, type tags, attribute suggestions, as well as attribute definitions with integrity constraints.

Generating knowledge is often a process with loose structure. Therefore, an extra category called knowledge-intensive processes was introduced to support knowledge work. These processes normally cannot be expressed by typical business process management systems because of their unpredictability and error-proneness. Recent researches worked out requirements and characteristics which are needed to provide a system supporting knowledge-intensive processes including the ability for modeling and abstracting.

The goal of this master's thesis is to combine a Hybrid Wiki with aspects of a system supporting knowledge-intensive aspects. The focus is not to create an application for modeling purposes but to create an application, which helps to structure and document processes for knowledge work based on the idea of tasks combined with the characteristics of the Hybrid Wiki

Student Project

ATTRIBUTES | TASKS

Fill out requirements

*Start date:* Jul 10, 2015

*End date:* Jul 15, 2015

*Progress:* 100%

*Sebis Contributor Agreement signed on* May 11, 2015

*Supervisor* Prof. Dr. Florian Matthes

*Type* Master's Thesis

Figure 5.2.: A screenshot of the center part of a wiki page showing the directive `scAttributes` by the box at the right.

### 5.2.3. Adaptions in scAngular

New functionalities are added to this library as well, due to the fact that `scAngular` is built for the main purpose of supporting communication between clients and SocioCortex. The added entities are basic entities which only needed support for CRUD operations. Therefore, the module `ngResource` was used as described in section 4.1.6. The complete extension of `scAngular` is illustrated in listing 5.1

The entities which are added to SocioCortex are explained in detail in section 6.2.1. There are the entities `Process`, `Stage`, `TaskDefinition`, `Task` and `Expertise`. Due to the scope of this thesis the only entities added are `Task` and `Expertise`. They were added to the submodule `scData` of `scAngular` which is representing the instance layer of SocioCortex.

```
var Task = $resource(
  getFullUrl(paths.tasks + "/:id"),
  { id: "@id" },
  {
    update: { method: "PUT" },
    getAttributes: {
        method: "GET",
        url: getFullUrl(paths.tasks + "/:id/" + paths.attributes),
        isArray: true
    }
  });

var Expertise = $resource(
  getFullUrl(paths.expertises + "/:id"),
  { id: "@id" },
  { update: { method: "PUT" } });
```

Listing 5.1: The complete extension of `scAngular`.

# 6. Task Support in SocioCortex

Collaborative knowledge management and knowledge generation are two features, which is supported in SocioCortex. Additionally to the concepts of the Hybrid Wiki which enabled evolving data structures like explained in section 2.1, it is also important to support the processes of knowledge creation.

This chapter explains the concepts that are implemented in the SocioCortex Core application.

## 6.1. Combine Tricia's conceptual model with aspects of Darwin

The Darwin Wiki is an approach to support Case Management, Knowledge Work and Virtual Crowdsourcing Communities [HKM15]. As the Darwin Wiki is a successor of the Hybrid Wiki its data model has certain similarities [HKM15]. SocioCortex is also a successor of the Hybrid Wiki [seb14]. Consequently, taking figure 2.1 and creating an overlay with figure 2.3 the existing Hybrid Wiki data model will be extended by 4 conceptual entities shown in figure 6.1: `Expertise`, `Task`, `TaskDefinition` and `Stage`.

In the Darwin Wiki `Task` and `TaskDefinition` are consequently extending the concept of instance and type layer. Furthermore, they are providing the possibility of evolving processes [HKM15]. `Stage` is also supporting the concepts of evolving processes however it is more related to support aggregation of tasks and defining different steps of completion. `Expertise` is for supporting a feature relating a collaborative approach of the Darwin Wiki. An expert is "a person who is very knowledgeable about or skilful in a particular area" [Oxf16a]. Assigning expertises to tasks is a way to tag a task for certain required knowledge. Completing task proves the knowledge of a user on this expertise. This means, a user can gain expertise by completing tasks. These informations were used in the Darwin Wiki for creating a reputation like system.

## 6.2. Implemented Concepts

The implemented concepts are based on the use cases stated in section 5.1. Therefore, the following sections are showing the implemented entities, roles and special concepts supporting the entities and roles. Explaining the structure of these as well as the relation between these will also be described in the following sections.

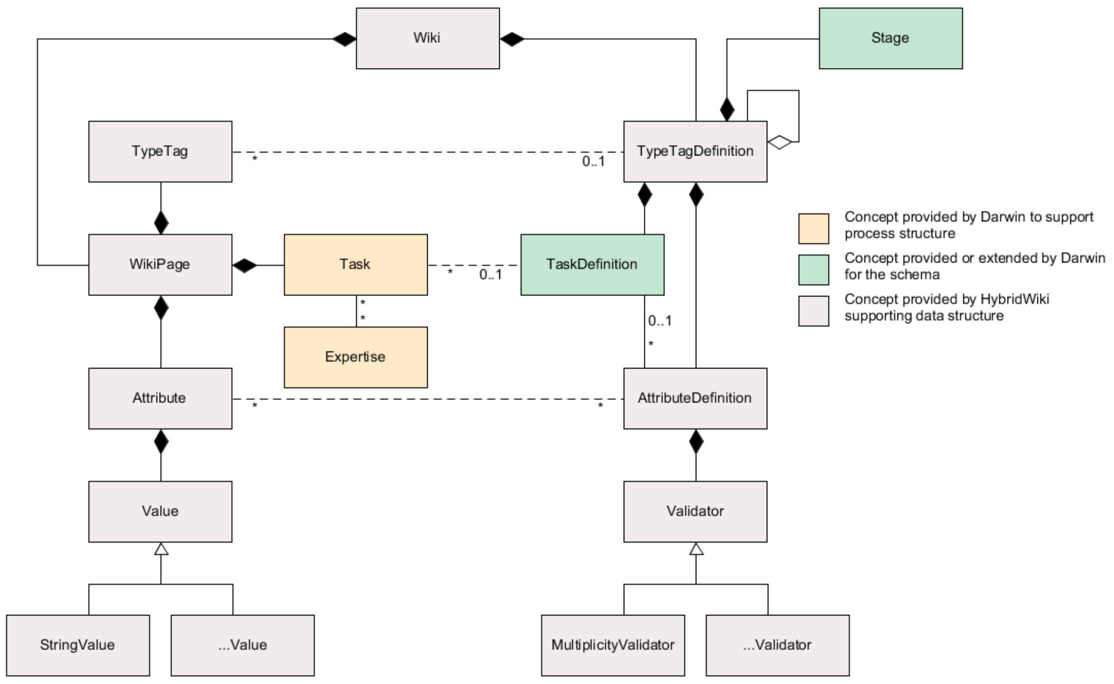


Figure 6.1.: The conceptual model of SocioCortex. The colored entities are added to the conceptual model as a result of this thesis. This model is a simplified version of the real implementation for providing a conceptual view for the results of overlaying figure 2.1 and figure 2.3.

### 6.2.1. Entities

The most important change to SocioCortex is the implementation of new entities. However, figure 6.1 shows the conceptual model of SocioCortex the implemented model is different. In figure 6.2 a more implementation-related model is represented. The newly added entities are described in the following. However, the entity `Rule` is mentioned in the diagram it is not part of the implementation, it shall provide an starting point for future implementations. Rules or sentries, how they are called in the context of CMMN, are the starting point for implementing more Case Management related capabilities, which are not in the scope of this thesis.

All introduced entities are subclasses of `PersistentEntity`. Therefore, they all have the attribute `id` already implemented.

#### Process

As a process is a series of actions it has to consist of single steps. `Process`, `Stage` and `TaskDefinition` are not creating a real composite pattern however using this structure enables enables to create series of processes as well as grouped steps of processes. Not being a real composite pattern means, that there is no single root node attached to the `TypeDefinition`. Instead the `TypeDefinition` itself is serving as root, which means it can provide multiple processes. Furthermore, the entities `TaskDefinition` and `Stage` have common attributes because `Process` is the super entity of these. The common attributes and therefore the attributes of `Process` are listed in the following:

**name**
Both, `Stage` and `TaskDefinition` are having a name. For the `TaskDefinition` it is important to note that its name will be inherited by the later instantiated page for its corresponding task.

**typeDefinition**
`Stage` and `TaskDefinition` can both be directly assigned to a `TypeDefinition`. Therefore, they both need a relation to the `TypeDefinition`.

Both of these attributes are needed in order to be able to create a new `Stage` or `TaskDefinition`.

#### Stage

As explained in the description of this section, the `Rule` is not implemented because it is out of scope for this thesis. The `Rule` entities are mainly depending on the `Stage` entity. A stage is the logical and semantical grouping of various processes. Using the concept of rules enables it to model more complex processes especially for depending processes. For example, one stage has to be finished, which means all processes within this stage have to be terminated in order to allow the the parent process to proceed.

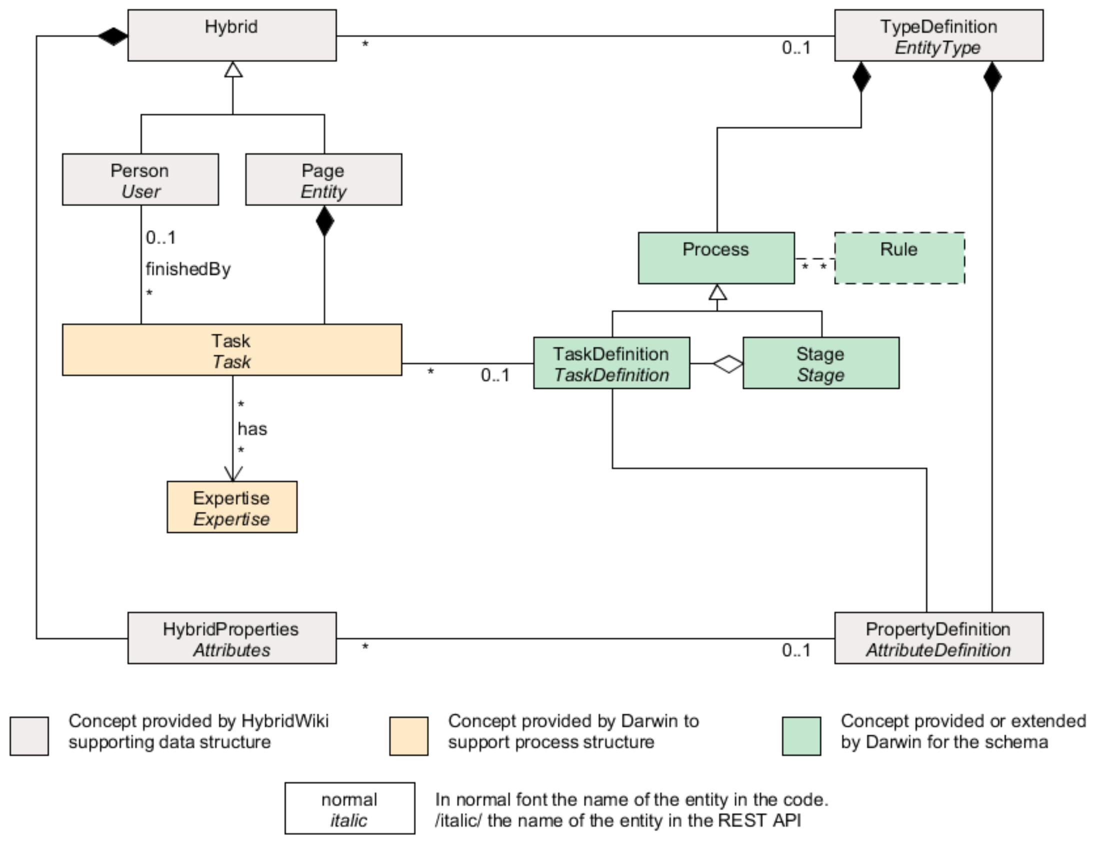

Hybrid
TypeDefinition
EntityType
*
0..1
Person
User
Page
Entity
0..1
finishedBy
*
Process
Rule
* *
Task
Task
TaskDefinition
TaskDefinition
Stage
Stage
*
0..1
*
has
*
Expertise
Expertise
HybridProperties
Attributes
PropertyDefinition
AttributeDefinition
*
0..1
Concept provided by HybridWiki supporting data structure
Concept provided by Darwin to support process structure
Concept provided or extended by Darwin for the schema
normal
italic
In normal font the name of the entity in the code.
/italic/ the name of the entity in the REST API

Figure 6.2.: Implementation-related conceptual model of SocioCortex. `Hybrid`, in the upper left corner, in the Hybrid Wiki was a mixin for every entity which was having a page to view like `Person`, `Page` or `File`. `Person` and `Page`, which are in the context of SocioCortex the entities `User` and `Entity`, are inheriting `Hybrid`. A task can only exist in the context of an page, therefore the relation is a composition. Furthermore, a `Task` can have assigned expertises as well itself can be assigned to a `TaskDefinition`. The `TaskDefinition` can be part of a `Stage`. Both, `TaskDefinition` and `Stage` are part of the `TypeDefinitions` processes. No part of the implementation but included for completeness are `Rules`, which can be added in future implementations to add more capabilities for Case Management.

The only attribute defined in `Stage` is the following:

**taskDefinitions**
: contains all assigned task of a stage. This attribute can contain no tasks, too. This means, a stage can be created by providing the informations needed for its super entity `Process`.

#### TaskDefinition

The definition of a task is serving as the blueprint of the task itself. Therefore, the task definition shall contain all information needed to complete a task. In the context of SocioCortex, this means to define the deliverables a user has to contribute in order to complete the task. Another function of a task's definition is to keep track of its instances. To provide all functionalities, the following attributes are implemented beside the attributes of the `Process` entity:

**propertyDefinitions**
: are needed do define deliverables for a task expressed by the task's attributes. Nevertheless, this attribute is not needed to be defined for a `TaskDefinition`. It is possible that a task is only using its progress state to detect its completeness.

**stage**
: as explained above a `TaskDefinition` can be part of a `Stage` to signal that it is related to another process but it does not have to. For creating a `TaskDefinition` this attribute is optional, too.

**tasks**
: this attribute of `TaskDefinition` is the collection which represents all instantiated `Tasks` of a particular `TaskDefinition`. When a `Page` is created the tasks will be initialized, too. Therefore, during runtime this attribute will get more instances assigned to it by created instances of the according `TypeDefinition` in form of `Pages`.

#### Expertise

As explained in section 6.1 an expertise is a concept to indicate the knowledge of a user in a specific domain. Therefore, expertises can be seen as simple text tags, that are attached to tasks to indicate which capabilities are needed to complete the task. It is represented as a list of keywords in the metadata of the task. Finishing a task, which means to complete it by either handing in all deliverables or by setting the progress to 100% the expertises of the task are assigned to the user who finished the task. The Darwin Wiki used expertises to provide an overview of the area of expertise for the users to encourage interaction between the users. If one user has a question he could look for another users who had a great number of expertises gained in the question's area in order to ask him for an answer.

`Expertises` are not attached to `TaskDefinitions`. However, a task may have the same attributes and name it is possible that a `Task` is adapted to needs of the current process and therefore needs different knowledge to be finished.

However, the possibilities for expertises are versatile the implemented model for an `Expertise` is rather simple:

**name**
: for identifying the expertise. This attribute has a constraint which allows only unique values for this attribute.

**tasks**
: which are related to a `Expertise` entity are listed in this attribute.

**Task**

Tasks are "a piece of work to be done or undertaken" [Oxf16c]. Therefore, `Task` is the entity the end-user will be using the most in the context of task support in SocioCortex. As stated in the definition it is representing a part of a work or process. This requires the task to provide several informations which are persisted in the `Task` entity. There are informations for a time-related planning, about the status of the task and the related persons who are working with the task. All use-cases declared in section 5.1 are related to this entity. Consequently, the attributes are all related to one of these. Due to the fact that Darwin is the based model the following attributes are very similar to Darwin's metadata of tasks:

**attributes**
: are the deliverables of a task. This means if every single attribute has at least one value the task will be finished unless the user decided to enter a customer value for the `progress` attribute. If the user decided to provide a custom value for `progress` he would be responsible for manually finishing the task.

  Basically, in Tricia this attribute is called `HybridProperty` as mentioned in figure 6.2

  The attribute `attributes` is important for the use cases *Task Execution* (5.1.3) and *Task Finishing* (5.1.4) because the attributes of a task are completed during the execution as well as the task will be finished if all attributes are completed. However, this attribute is important for finishing a task it is not mandatory for creating a new one because editing the `progress` attribute directly is a second way to finish a task.

**begin**
: declares a date at which the task begins. This is important for *Task Planning* (5.1.2) in order to display the chronological order of the tasks in the Gantt chart and for *Process Adaption* (5.1.5) in order to alter the process. It is also important for *Task Execution* (5.1.3) to indicate when a task should start for keeping the schedule of the process.

**end**
is also mostly important for *Task Planning* (5.1.2) in order to plan the chronological order and for *Process Adaption* (5.1.5) for altering the process like described in the attribute `begin`. Another functionality of this attribute is used by *Task Notifications* (5.1.6). This use case is showing the tasks which should be done according to their end date which is defined by this attribute.

**expertises**
is already explained in the section of the entity `Expertise` as well as in section 6.1. This attribute contains areas of knowledge which are needed to be used in order to fulfill a certain task. This attribute is needed for *Task Planning* (5.1.2) and *Task Execution* (5.1.3) to know if a person is suitable for a task.

**finishedAt**
is simply the date a user finished a task. This date is set when the `progress` attribute is going to be 100%. This can be done by either setting the `progress` attribute directly to 100% or by filling out all attached attributes as long as `isProgressCalculated` is still `false`. It is used for helping in the *Task Planning* (5.1.2) and *Process Adaption* (5.1.5) to adjust the schedule of the overall process. Consequently, another addressed use case is *Task Finishing* (5.1.4) because it is triggered when a task's state is set to finished.

**finishedBy**
represents the user who finished a task and is defined by the same conditions as `finishedAt`. This attribute was added in order to be able to create further functionalities for the community aspect of SocioCortex. As well as `finishedAt` this attribute is related with *Task Finishing* (5.1.4) because it is triggered when the task is finished, too.

**isInconsistent**
is a boolean attribute, which is set by a change listener of both, the begin as well as the end attribute and can not be set manually. This change listener is checking the integrity using the task hierarchy, which is similar to the Darwin Wiki's hierarchy described in section 2.2.2. Like the other time-related attributes it is needed for the use cases focusing the process' schedule, namely: *Task Planning* (5.1.2) and *Process Adaption* (5.1.5).

**isOverdue**
is another boolean attribute like `isInconsistent` and it can not be set manually, too. It is set by a periodically running server task to determine if the end date of a task is beyond the current date. If this attribute results to `true` the overdue task will be showed as noted in *Task Notifications* (5.1.6).

**isProgressCalculated**
indicates if the `progress` property was ever manually overwritten or not. If a user

overwrites the `progress` property a change listener will be activated which turns this attribute from `false` to `true`. This attribute addresses the use case *Process Adaption* (5.1.5) because it enables to finish a task manually without the need of providing all deliverables.

**name**
is the name of the task. If the task is derived from a `TaskDefinition` this name will be the same as the `TaskDefinition`'s name. The name shall be an indicator what the task is about and what has to be done. If a new task is created without having a `TaskDefinition` this attribute will be mandatory.

**page**
represents the containing page of a `Task` entity and therefore the concrete context of a task. As a task can not exist without a page this attribute is mandatory if a new task is created.

**progress**
indicates the current status of a task. If this values is not edited manually it will show the process calculated using the grade of fulfillment of the task's attributes. This attribute has the constraints of being in a range between 0 and 100 in order to represent a percentage value. It is especially important for *Process Adaption* (5.1.5) because it empowers the user to keep track of a process. Another important use case related to the `progress` attribute is *Task Planning* (5.1.2) because it enables to use the Gantt chart as a timetable. The third important use case is *Task Finishing* (5.1.4) because the `progress` attribute is responsible to finish the task either by manually be set by the user or by automatically be calculated.

**skipped**
is a boolean attribute. If a user decides to not finish a task he is able to skip it. If the user skipped the task this boolean will be `true`. In the context of KIPs it is necessary to have a mechanism to skip processes if a exception is occurring. This relates this attribute to *Process Adaption* (5.1.5) in handling unforeseen events.

**skippedAt**
is used to keep track of the process status. By clicking on the skip button of a task the time will be noted to support transparent processes.

**skippedBy**
is used similar to `skippedAt` unless this time the user will be tracked.

**taskDefinition**
is another attribute which provides the context of blueprints and empowering the users of SocioCortex to evolve processes. The concept of the definition of tasks is explained in the entity's description of `TaskDefinition` as well as in section 6.1 and chapter 2.

### 6.2.2. Constraints added to the model

In the persistence layer of Tricia and SocioCortex `Roles` are associations between entities. Normally `Roles` are simple attributes added to the entities like the attribute `taskDefinition` in `Task`. There are two main classes supporting associations: `OneRole` and `ManyRole`. Both of these are implementing the interface `Role`. Therefore, these special attributes are able to map any multiplicities like one-to-one, one-to-many and many-to-many like the one-to-many example in listing 6.1. Unfortunately, the model itself does not disallow all possible side effects. For avoiding these, several constraints were added to some of the relations.

Most of these problems are potentially compromising data integrity. This is occurring in most times by an entity, which is related to another using two disjunct paths. In the following these problems are explained.

```java
import de.infoasset.platform.orm.ManyRole;
import de.infoasset.platform.orm.OneRole;

public class Task extends PersistentEntity {
  // [...]

  public final OneRole<TaskDefinition> taskDefinition = new OneRole<
      TaskDefinition>() {
    protected ManyRole<Task> oppositeRole() {
      return TaskDefinition.SCHEMA.prototype().tasks;
    };
  };

  // [...]
}
```

Listing 6.1: An example for an implemented role. Using an anonymous class the role is added to the entities model. In this anonynmous class a method called `oppositeRole` is overridden, which is pointing to the counterpart of the association, namely the attribute `tasks` of `TaskDefinition`. Using Java's reflection package this role is found by the ORM layer of SocioCortex

#### TaskDefinition – Stage

Unfortunately, by design it is allowed for `Stages` and `TaskDefinitions` to be assigned to different `TypeDefinitions`. To disable this effect, a constraint was added to `TaskDefinition`. If the `Stage` assigned to a `TaskDefinition` has a different `TypeDefinition` a exception will be thrown, which roll backs the persisting request.

#### TaskDefinition – PropertyDefinition

The same problem existing for TaskDefinition – Stage exists for `TaskDefinition` and `PropertyDefinition`. These two entities can also be assigned to two different `TypeDefinitions`.

#### TaskDefinition – Task

The third `Role` defined in `TaskDefinition` is also able to have a different `TypeDefinition`. This time it is not directly related but by the relation of `Page` to `TypeDefinition` the link exists and has to be assured of being a valid association.

#### Task – Attribute

A wiki page has attributes and tasks are having attributes, too. The constraint is that the task's attributes are a subset of the wiki page's one. Therefore, the constraint had to be added in the application layer and could not be restricted by design.

#### Circular References within Pages

In particular there is one special change listener which is added to the model of SocioCortex. Due to the fact that attributes of wiki pages are able to reference other wiki pages it is possible to create circular references. Consequently, there is a change listener firing every time the values of `Attributes` are set. If it is containing a page it is following the hierarchy to find any bad references.

### 6.2.3. Automatically initializing Attributes

In Tricia attributes of wiki pages were not initialized when a new wiki page was created, which means the definitions of the attributes were not instantiated. This enabled attributes which were not having a value and which's definition was deleted after creating the page to disappear from the end-users view. This feature was changed in order to initialize the attributes during the creation of a change. The decision was made based on the decision of Darwin, which supported the same behavior and because of the argument that requirements are able to change over time however they may have made sense in the past and should therefore still be included in these pages.

### 6.2.4. OverdueTaskTimer & OverdueTaskJob

SocioCortex is also providing a concept for periodic events called `TimerTask`. Therefore, the `TaskManager` is used to register new tasks. Basically, the `TaskManager` is a wrapper for an Java `ExecuterService` which is started at the platform initialization process. Registering tasks is similar to the registration of entities in the SocioCortex REST API and the persistence layer. An instance of `AbstractTimerTask` is placed within the method `TaskManager.scheduleAllTasks`. On this instance method `schedule` will be called which starts the timer of the task.

The `OverdueTaskTimer` is a periodically running task which is started once a day at midnight. Its task is to find all `Task` entities which are overdue by the beginning of the new day. This is done by using a subclass of `TaskTimer` called `OnceADayAtSpecificTimeTimerTask`. The abstract method which is overriden by the `OverdueTaskTimer` specifies the hour at which the job has to be executed.

## 6.3. REST API

The REST API of SocioCortex has its own documentation[1]. Basically the REST interface is mirroring the attributes described in section 6.2.1. All CRUD operations which are needed to be able to implement the use-cases of section 5.1 are implemented. Therefore, every entity described in section 6.2.1 has its own path to be accessed.

**`/api/v1/tasks`**

**`[GET]`** retrieves a list of `Tasks`.

**`[POST]`** creates a new `Task`.

**`/api/v1/tasks/{taskId}`**

**`[GET]`** retrieves a specific `Task` with the id `taskId`

**`[PUT]`** updates a specific `Task` with the id `taskId`

**`[DELETE]`** deletes the `Task` with the id `taskId`

**`/api/v1/tasks/{taskId}/expertises`**

**`[GET]`** retrieves the expertises of the `Task` with the id `taskId`

**`[POST]`** adds an expertise to the `Task` with the id `taskId`

**`/api/v1/expertises`**

**`[GET]`** retrieves a list of all `Expertises`

**`[POST]`** creates a new `Expertise`

**`/api/v1/expertises/{expertiseId}`**

[1] `http://www.sociocortex.com/documentation/`, accessed on 10.01.2016

**[GET]** retrieves the `Expertise` with the id `expertiseId`

**[DELETE]** deletes the `Expertise` with the id `expertiseId`

**`/api/v1/taskDefinitions`**

**[GET]** retrieves a list of all `TaskDefinitions`

**[POST]** creates a new `TaskDefinition`

**`/api/v1/taskDefinitions/{taskDefinitionId}`**

**[GET]** retrieves the `TaskDefinition` with the id `taskDefinitionId`

**[PUT]** updates the `TaskDefinition` with the id `taskDefinitionId`

**[DELETE]** deletes the `TaskDefinition` with the id `taskDefinitionId`

**`/api/v1/taskDefinitions/{taskDefinitionId}/attributeDefinitions`**

**[GET]** retrieves a list of the `AttributeDefinitions` assigned to the `TaskDefinition` with the id `taskDefinitionId`

**`/api/v1/stages`**

**[GET]** retrieves a list of all `Stages`

**[POST]** creates a new `Stage`

**`/api/v1/stages/{stageId}`**

**[GET]** retrieves the `Stage` with the id `stageId`

**[PUT]** updates the `Stage` with the id `stageId`

**[DELETE]** deletes the `Stage` with the id `stageId`

**`/api/v1/stages/{stageId}/taskDefinitions`**

**[GET]** retrieves all `TaskDefinitions` of the `Stage` with the id `stageId`

# Part III.

# Conclusion

# 7. Summary & Conclusion

This chapter is providing a summary of this thesis followed by a conclusion. The last section conveys an outlook for further developments.

## 7.1. Summary

The objective of thesis was to create a basic task support in SocioCortex including features of the former implemented Darwin Wiki. To accomplish this, the first step was to investigate the Hybrid Wiki which is the predecessor of SocioCortex. The found concepts of Type Tags and Attributes were discussed and related to the Darwin Wiki in chapter 2.

Similar to SocioCortex, Darwin Wiki is a successor of Hybrid Wiki. The difference of Hybrid Wiki and Darwin Wiki was detected in section 2.2.1. While the first one is focusing on data and the development of data structures, the latter one is centralized on figuring out which processes are responsible for the evolved data structures and how these processes can be structured. Both approaches have in common to empower the end-user for contributing to knowledge work and knowledge intensive processes.

After comparing related approaches in chapter 3 an overview for already existing projects related to SocioCortex was given in section 3.5. In addition, the used technologies were introduced in chapter 4. Especially focused was scAngular in section 4.1.6 because it is used as the connecting part of the SocioCortex Core Application and the SocioCortex Generic Client.

The basic architecture of the Socio Cortex Generic Client is another deliverable of this thesis. Therefore, the basic functionality for a client application was introduced in chapter 5, whereby the discovered use cases are described in section 5.1. All of them are related to tasks due to the task-centered scope of the thesis. The resulted approach implements functionalities for supporting knowledge intensive processes based on the Darwin Wiki which's evaluated concepts provide methodologies to develop a flexible process structure over time by using unstructured data. After introducing the concepts of the Generic Client, the implementation-related structure is described in section 5.2.

The second deliverable of this thesis was to include task support in the existing SocioCortex Core Application. The emerged results of chapter 2 were used to extend the conceptual data model of SocioCortex with new entities. Therefore, the conceptual model of SocioCortex, which is derived from the Hybrid Wiki, was compared with the model of the Darwin Wiki. The resulted extended model is explained in chapter 6 along with added entities which are described in detail in section 6.2.1 including the usages of its attributes.

Basically, five new entities were introduced: the entities `Process`, `Stage` and `TaskDefinition` are used for defining processes whereas `Task` and `Expertise` are used to execute a single process. The design of these entities is consequently continuing the concepts of the Darwin Wiki and Hybrid Wiki. That is, to use unstructured data provided by end-users for developing new structures. These can then be used for further guiding the end-users or for evaluating business domain model concepts.

In general, SocioCortex was extended by a basic task-supporting concept by investigating former approaches and comparing their conceptual models. The provided solution enables the use of SocioCortex as a task-driven collaborative knowledge generation tool. Whereas not all implemented concepts of SocioCortex are used in the Generic Client, the client's structure enables to use it as a basis for further implementations.

## 7.2. Conclusion

Angular Material as front end framework provided solid components for implementing the client. However, the focus of Material Design is on mobile clients, creating a desktop client is possible with restrictions. Material Design is made for simple designs using only a few components. The SocioCortex Generic Client combines various features in one view like in figure 5.1: the navigation bar at the top containing general settings, the attributes representation at the right containing possibilities for process adaptions and the navigation bar on the left of the screen. Therefore, adjustments were made regarding spacings and sizes of the components.

SocioCortex' implementation of a persistence layer provided unconventional solutions for the needs of implementing the task-centered extensions comparing to JPA or Hibernate. The use of anonymous classes in conjunction with Java's reflection API empowers the developer to write associations and assertions directly into the entity's class. However, writing a significant number of anonymous classes results in a certain amount of boilerplate code which can lead to lack of understandability.

Using the provided documentation of SocioCortex' ORM layer it was possible to create the needed structures. The capabilities of adapting persisted database structures provided a fluent implementation process regarding to changing requirements.

## 7.3. Outlook

Summing up, a basic task support was implemented. However, there are concepts left which are introduced in the Darwin Wiki. The entity `Rule` of the conceptual model presented in figure 6.2 on page 58 is not implemented because of not being in the scope of this thesis.

If there are too many tasks displayed on a page, end-users may be overwhelmed by the choice which task should be executed next. Therefore, the concept of rules is introduced. The main purpose of rules is to create logical links between tasks and stages [Hau15]. They shall help end-users to choose the next task to be completed. In the Darwin Wiki rules are implemented by only displaying a current subset of tasks whereby every task has to be finished in order for displaying new tasks. This concept, of predecessors producing an output and successor consuming it, is called the producer and consumer pattern and is often used in KIPs [Hau15]. There are derived pattern types, too, using multiple producers or multiple consumers.

Another concept used in Darwin is to motivate users in participating to the wiki by using social principles. Social principles are gathered concepts by investigating successful online communities [KRK11]. One example for a principle added to Darwin is a user rating based on the user's activity. Therefore, two visualizations are introduced. The first one is a pie chart representing the user's expertises. However, there is no exact number how many expertises the user gathered by finishing tasks, it is presenting the user's emphasis. The second visualization is representing the activity of the user in general. It is a bar with a triangle shaped indicator. The bar is representing a percentage from 0% to 100%. The triangle is pointing to an percentage which is calculated for a specific user $X$ by the following equation:

$$\frac{\text{\# of users who finished less tasks than user X}}{\text{\# of all users} - 1}$$

Motivation is generated by these visualizations because of different addressed principles [Hau15]. A first example is a user who needs help. Based on the reputation he is able to find a user who could offer support. Second, users get frequent feedback about the personal focus and can compare the provided information with their personal goals. As a third example: users get motivated due to the implicit competition which is implied by seeing the personal rank on the bar chart.

# Part IV.

# Appendix

# 8. Adapted Metamodels relevant for SocioCortex

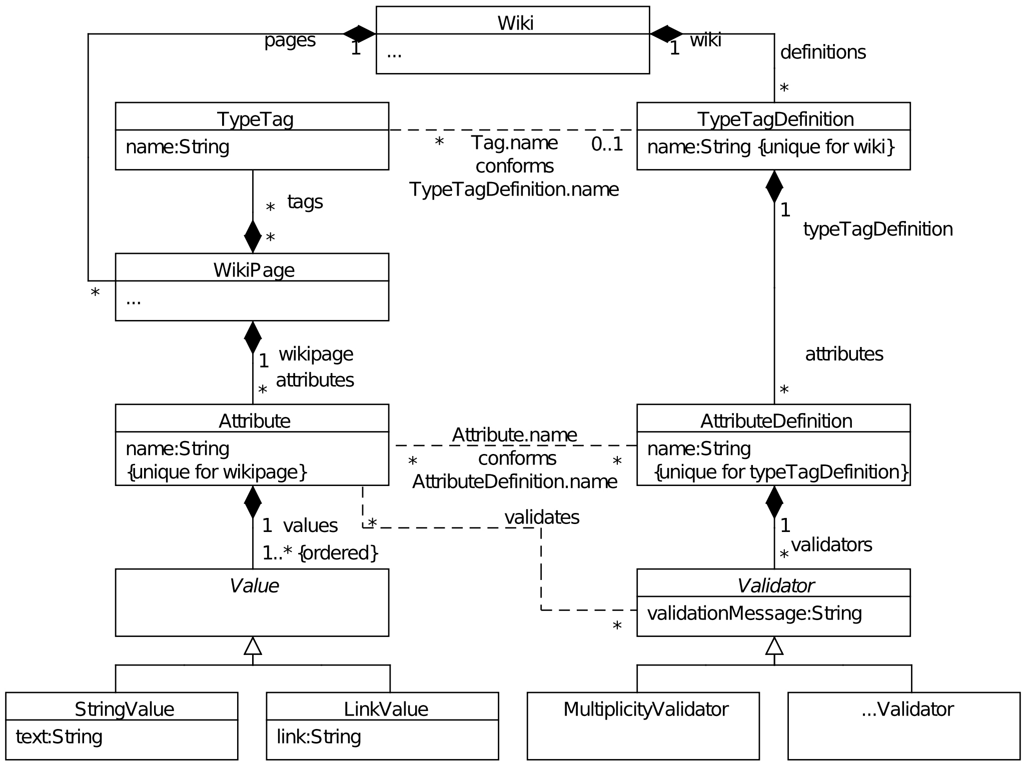


Figure 8.1.: The data model for Hybrid Wikis [MNS11]

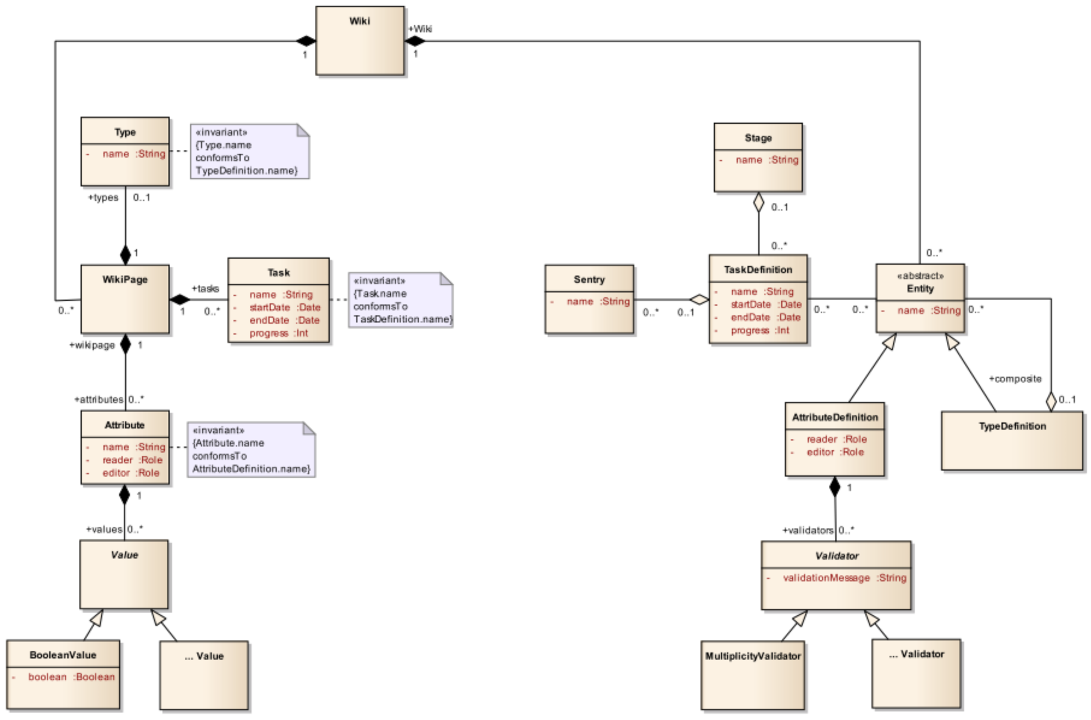


Figure 8.2.: The conceptual data model for Darwin Wikis [HKM15].

# Bibliography

[AL07] AUER, Sören ; LEHMANN, Jens: What Have Innsbruck and Leipzig in Common? Extracting Semantics from Wiki Content. Version: 2007. http://dx.doi.org/10.1007/978-3-540-72667-8{_}36. In: FRANCONI, Enrico (Hrsg.) ; KIFER, Michael (Hrsg.) ; MAY, Wolfgang (Hrsg.): *The semantic web: research and applications* Bd. 4519. Berlin : Springer, 2007. – DOI 10.1007/978–3–540–72667–8_36. – ISBN 978–3–540–72666–1, S. 503–517

[Ang15a] ANGULARJS: *AngularJS / Developer Guide / Conceptual Overview*. https://docs.angularjs.org/guide/concepts. Version: 2015

[Ang15b] ANGULARJS: *$resource: service in module ngResource*. https://docs.angularjs.org/api/ngResource/service/$resource. Version: 2015

[Bec04] BECKETT, Dave: *RDF/XML Syntax Specification (Revised)*. http://www.w3.org/TR/REC-rdf-syntax/. Version: 2004

[BG14] BRICKLEY, Dan ; GUHA, R. V.: *RDF Schema 1.1*. http://www.w3.org/TR/rdf-schema/. Version: 2014

[BLHL01] BERNERS-LEE, Tim ; HENDLER, James ; LASSILA, Ora: The semantic web. In: *Scientific american* 284 (2001), Nr. 5, S. 28–37

[BSVW12] BRY, François ; SCHAFFERT, Sebastian ; VRANDEČIĆ, Denny ; WEIAND, Klara: Semantic Wikis: Approaches, Applications, and Perspectives. Version: 2012. http://dx.doi.org/10.1007/978-3-642-33158-9{_}9. In: EITER, Thomas (Hrsg.) ; KRENNWALLNER, Thomas (Hrsg.): *Reasoning web* Bd. 7487. Berlin : Springer, 2012. – DOI 10.1007/978–3–642–33158–9_9. – ISBN 978–3–642–33157–2, S. 329–369

[Bür15] BÜRGIN, Patrick: *Design and Prototypical Implementation of a Dashboard System for Visualizing Semi-Structured Data in a Traceable Way*. München, Technische Universität München, Master's Thesis, 2015. https://wwwmatthes.in.tum.de/pages/16qwn6mc8kei2/Master-s-Thesis-Patrick-Buergin

[CE08] CHAMBERS, Chris ; ERWIG, Martin: Dimension inference in spreadsheets. In: BOTTONI, Paolo (Hrsg.) ; ROSSON, Mary B. (Hrsg.) ; MINAS, Mark (Hrsg.): *IEEE Symposium on Visual Languages and Human-Centric Computing*. Piscataway, NJ : IEEE, 2008. – ISBN 978–1–4244–2528–0, S. 123–130

[CL02] CHOI, Byounggu ; LEE, Heeseok: Knowledge management strategy and its link to knowledge creation process. In: *Expert Systems with Applications* 23 (2002), Nr. 3, S. 173–187. http://dx.doi.org/10.1016/S0957-4174(02)00038-6. – DOI 10.1016/S0957–4174(02)00038–6. – ISSN 0957–4174

[Dic15] DICKEY, Jeff: *Write modern web apps with the MEAN stack: Mongo, Express, AngularJS, and Node.jss*. San Francisco, Calif. : Peachpit Press, 2015 (Develop and design). – ISBN 9780133930153

[GMR+15] GIL, Yolanda ; MICHEL, Felix ; RATNAKAR, Varun ; HAUDER, Matheus ; DUFFY, Christopher ; DUGAN, Hilary ; HANSON, Paul: A Task-Centered Framework for Computationally-Grounded Science Collaborations. In: *11th IEEE International Conference on e-Science*, 2015, S. 352–361

[Goo15a] GOOGLE: *Material Design: Introduction*. http://www.google.com/design/spec/material-design/introduction.html. Version: 2015

[Goo15b] GOOGLE: *What is material? Elevation and shadows*. http://www.google.com/design/spec/what-is-material/elevation-shadows.html#elevation-shadows-elevation-android-. Version: 2015

[Hau15] HAUDER, Matheus: *Empowering End-Users to Collaboratively Structure Knowledge-Intensive Processes*. München, Technische Universität München, PhD Thesis, 2015

[HKM15] HAUDER, Matheus ; KAZMAN, Rick ; MATTHES, Florian: Empowering End-Users to Collaboratively Structure Processes for Knowledge Work. Version: 2015. http://dx.doi.org/10.1007/978-3-319-19027-3{_}17. In: ABRAMOWICZ, Witold (Hrsg.): *Business Information Systems* Bd. 208. Cham : Springer International Publishing, 2015. – DOI 10.1007/978–3–319–19027–3_17. – ISBN 978–3–319–19026–6, S. 207–219

[HKRS08] HITZLER, Pascal ; KRÖTZSCH, Markus ; RUDOLPH, Sebastian ; SURE, York: *Semantic Web: Grundlagen*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2008 (eXamen.press). http://dx.doi.org/10.1007/978-3-540-33994-6. http://dx.doi.org/10.1007/978-3-540-33994-6. – ISBN 9783540339946

[HLS05] HAAKE, Anja ; LUKOSCH, Stephan ; SCHÜMMER, Till: Wiki-templates: adding structure support to wikis on demand. In: RIEHLE, Dirk (Hrsg.): *Proceedings of the 2005 international symposium on Wikis*. New York, NY : ACM, 2005. – ISBN 1–59593–111–2, S. 41–51

[HPS14] HAYES, Patrick J. ; PATEL-SCHNEIDER, Peter F.: *RDF 1.1 Semantics: W3C Recommendation*. http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/. Version: 2014

[HT08] HAPPEL, Hans-Jörg ; TREITZ, Marius: Proliferation in Enterprise Wikis. In: *8th International Conference on the Design of Cooperative Systems*, 2008, 123–129

[Kat15] KATENBRINK, Florian: *Optimizing the User Experience of a Social Content Manager for Casual Users*. München, Technische Universität München, Bachelor's Thesis, 15.08.2015. https://wwwmatthes.in.tum.de/file/zvvzsqkbo8vt/Sebis-Public-Website/-/Bachelor-s-Thesis-Florian-Katenbrink/Thesis.pdf

[KRK11] KRAUT, Robert E. ; RESNICK, Paul ; KIESLER, Sara: *Building successful online communities: Evidence-based social design*. Cambridge, Mass : MIT Press, 2011 http://lib.myilibrary.com/detail.asp?id=359469. – ISBN 978–0262016575

[KS08] KUHN, Tobias ; SCHWITTER, Rolf: Writing support for controlled natural languages. In: STOKES, Nicola (Hrsg.) ; POWERS, David (Hrsg.): *Proceedings of ALTA* Bd. 6, 2008, S. 46–54

[Kuh09] KUHN, Tobias: How Controlled English can Improve Semantic Wikis. In: LANGE, Christoph (Hrsg.) ; SCHAFFERT, Sebastian (Hrsg.) ; SKAF-MOLLI, Hala (Hrsg.) ; VÖLKEL, Max (Hrsg.): *4th Semantic Wiki Workshop (SemWiki 2009) at the 6th European Semantic Web Conference (ESWC 2009)* Bd. 464, CEUR-WS.org, 2009 (CEUR Workshop Proceedings)

[KVV05] KRÖTZSCH, Markus ; VRANDEČIĆ, Denny ; VÖLKEL, Max: Wikipedia and the Semantic Web - The Missing Links. In: *Proceedings of the WikiMania 2005*, 2005

[KVV06] KRÖTZSCH, Markus ; VRANDEČIĆ, Denny ; VÖLKEL, Max: Semantic MediaWiki. Version: 2006. http://dx.doi.org/10.1007/11926078{_}68. In: HUTCHISON, David (Hrsg.) ; KANADE, Takeo (Hrsg.) ; KITTLER, Josef (Hrsg.) ; KLEINBERG, Jon M. (Hrsg.) ; MATTERN, Friedemann (Hrsg.) ; MITCHELL, John C. (Hrsg.) ; NAOR, Moni (Hrsg.) ; NIERSTRASZ, Oscar (Hrsg.) ; PANDU RANGAN, C. (Hrsg.) ; STEFFEN, Bernhard (Hrsg.) ; SUDAN, Madhu (Hrsg.) ; TERZOPOULOS, Demetri (Hrsg.) ; TYGAR, Dough (Hrsg.) ; VARDI, Moshe Y. (Hrsg.) ; WEIKUM, Gerhard (Hrsg.) ; CRUZ, Isabel (Hrsg.) ; DECKER, Stefan (Hrsg.) ; ALLEMANG, Dean (Hrsg.) ; PREIST, Chris (Hrsg.) ; SCHWABE, Daniel (Hrsg.) ; MIKA, Peter (Hrsg.) ; USCHOLD, Mike (Hrsg.) ; AROYO, Lora M. (Hrsg.): *The Semantic Web - ISWC 2006* Bd. 4273. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006. – DOI 10.1007/11926078_68. – ISBN 978–3–540–49029–6, S. 935–942

[Mei15] MEISSNER, Alexander: *Design and Prototypical Implementation of a Web based Spreadsheet System for Managing and Analyzing Semi-structured Data*. München, Technische Universität München, Master's Thesis, 2015. https://wwwmatthes.in.tum.de/pages/jnl8lsds3gz8/Master-s-Thesis-Alexander-Meissner

[MGRH15] MICHEL, Felix ; GIL, Yolanda ; RATNAKAR, Varun ; HAUDER, Matheus: A Virtual Crowdsourcing Community for Open Collaboration in Science Processes. In: *AMCIS 2015 Proceedings* (2015). http://aisel.aisnet.org/amcis2015/VirtualComm/GeneralPresentations/3

[MHV13] MARIN, Mike A. ; HULL, Richard ; VACULÍN, Roman: Data Centric BPM and the Emerging Case Management Standard: A Short Survey. Version: 2013. http://dx.doi.org/10.1007/978-3-642-36285-9{_}4. In: LA ROSA, Marcello (Hrsg.) ; SOFFER, Pnina (Hrsg.): *Business Process Management Workshops* Bd. 132. Springer Berlin Heidelberg, 2013. – DOI 10.1007/978–3–642–36285–9_4. – ISBN 978–3–642–36284–2, S. 24–30

[MLV14] MARIN, Mike A. ; LOTRIET, Hugo ; VAN DER POLL, John A.: Measuring Method Complexity of the Case Management Modeling and Notation (CMMN). Version: 2014. http://dx.doi.org/10.1145/2664591.2664608. In: VAN DEVENTER, J. P. (Hrsg.) ; VILLIERS, C. d. (Hrsg.) ; VAN DER MERWE, Alta (Hrsg.): *Proceedings of the Southern African Institute for Computer Scientist and Information Technologists Annual Conference 2014 on SAICSIT 2014 Empowered by Technology // Proceedings, SAICSIT2014*. Centurion, South Africa : ACM and Association for Computing Machinery, 2014 (ACM international conference proceedings series). – DOI 10.1145/2664591.2664608. – ISBN 978–1–4503–3246–0, S. 209–216

[MNS11] MATTHES, Florian ; NEUBERT, Christian ; STEINHOFF, Alexander: Hybrid Wikis: Empowering Users to Collaboratively Structure Information Software and Data Technologies, Volume 1, Seville, Spain, 18-21 July, 2011. In: ESCALONA, María José (Hrsg.) ; SHISHKOV, Boris (Hrsg.) ; CORDEIRO, José (Hrsg.): *ICSOFT 2011 - Proceedings of the 6th International Conference on Software and Data Technologies, Volume 1, Seville, Spain, 18-21 July, 2011*, SciTePress, 2011. – ISBN 978–989–8425–76–8, S. 250–259

[MV09] MCGUINNESS, Deborah L. ; VAN HARMELEN, Frank: *OWL Web Ontology Language: Overview*. http://www.w3.org/TR/owl-features/. Version: 2009

[Neu12] NEUBERT, Christian A.: *Facilitating Emergent and Adaptive Information Structures in Enterprise 2.0 Platforms*. München, Technische Universität München, PhD Thesis, 2012. https://wwwmatthes.in.tum.de/pages/51wnlrspsn3n/Ne12-Facilitating-Emergent-and-Adaptive-Information-Structures-in-Enterprise-2.0-Platforms

[Obj14] OBJECT MANAGEMENT GROUP ; OBJECT MANAGEMENT GROUP (Hrsg.): *Case Management Model And Notation (CMMN)*. http://www.omg.org/spec/CMMN/1.0/. Version: 05.05.2014 (1.0)

[Oxf16a] OXFORD DICTIONARIES: *expert: definition of expert in English from the Oxford dictionary*. http://www.oxforddictionaries.com/definition/english/expert. Version: 2016

[Oxf16b] OXFORD DICTIONARIES: *process: definition of process in English from the Oxford dictionary*. http://www.oxforddictionaries.com/definition/english/process. Version: 2016

[Oxf16c] OXFORD DICTIONARIES: *task: definition of task in English from the Oxford dictionary*. http://www.oxforddictionaries.com/definition/english/task. Version: 2016

[PNJB] PFISTERER, Frederik ; NITSCHE, Markus ; JAMESON, Anthony ; BARBU, Catalin: User-Centered Design and Evaluation of Interface Enhancements to the Semantic MediaWiki. https://www.researchgate.net/profile/Markus_Nitsche/publication/230640255_User-Centered_Design_and_Evaluation_of_Interface_Enhancements_to_the_Semantic_MediaWiki/links/0912f5024d22bafee1000000.pdf. In: *Workshop on Semantic Web User Interaction at CHI 2008*

[PR00] PEMBERTON, J. D. ; ROBSON, A. J.: Spreadsheets in business. In: *Industrial Management & Data Systems* 100 (2000), Nr. 8, 379–388. http://dx.doi.org/10.1108/02635570010353938. – DOI 10.1108/02635570010353938

[PS08] PRUD'HOMMEAUX, Eric ; SEABORNE, Andy: *SPARQL Query Language for RDF*. http://www.w3.org/TR/rdf-sparql-query/. Version: 2008

[PV06] PESIC, M. ; VAN DER AALST, Wil M. P.: A Declarative Approach for Flexible Business Processes Management. Version: 2006. http://dx.doi.org/10.1007/11837862{_}18. In: EDER, Johann (Hrsg.) ; DUSTDAR, Schahram (Hrsg.): *Business Process Management Workshops* Bd. 4103. Springer Berlin Heidelberg, 2006. – DOI 10.1007/11837862_18. – ISBN 978–3–540–38444–1, S. 169–180

[RRV03] REIJERS, H. A. ; RIGTER, J. H. M. ; VAN DER AALST, Wil M. P.: The Case Handling Case. In: *International Journal of Cooperative Information Systems* 12 (2003), Nr. 03, S. 365–391. http://dx.doi.org/10.1142/S0218843003000784. – DOI 10.1142/S0218843003000784. – ISSN 0218–8430

[SBBK07] SCHAFFERT, Sebastian ; BRY, François ; BAUMEISTER, Joachim ; KIESEL, Malte: Semantic Wiki. In: *Informatik-Spektrum* 30 (2007), Nr. 6, S. 434–439. http://dx.doi.org/10.1007/s00287-007-0195-z. – DOI 10.1007/s00287–007–0195–z. – ISSN 0170–6012

[Sch06] SCHAFFERT, Sebastian: IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In: REDDY, Sumitra M. (Hrsg.): *15th IEEE International*

*Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Piscataway, NJ : IEEE, 2006. – ISBN 0–7695–2623–3, S. 388–396

[Sch15] SCHALKAMP, Jan: *An Integrated Modeling Environment for Social Applications*. München, Technische Universität München, Master's Thesis, 2015. https://wwwmatthes.in.tum.de/pages/qswfufi5g6yg/Master-s-Thesis-Jan-Schalkamp

[seb14] SEBIS ; SOFTWARE ENGINEERING FOR BUSINESS INFORMATION SYSTEMS (Hrsg.): *SocioCortex*. 2014

[SM95] STRONG, Diane M. ; MILLER, Steven M.: Exceptions and exception handling in computerized information processes. In: *ACM Transactions on Information Systems* 13 (1995), Nr. 2, S. 206–233. http://dx.doi.org/10.1145/201040.201049. – DOI 10.1145/201040.201049. – ISSN 10468188

[SSSF09] SINT, Rolf ; SCHAFFERT, Sebastian ; STROKA, Stephanie ; FERSTL, Roland: Combining unstructured, fully structured and semi-structured information in semantic wikis. In: LANGE, Christoph (Hrsg.) ; SCHAFFERT, Sebastian (Hrsg.) ; SKAF-MOLLI, Hala (Hrsg.) ; VÖLKEL, Max (Hrsg.): *4th Semantic Wiki Workshop (SemWiki 2009) at the 6th European Semantic Web Conference (ESWC 2009)* Bd. 464, CEUR-WS.org, 2009 (CEUR Workshop Proceedings), S. 73

[Tei13] TEIXEIRA, Pedro: *Professional Node.js: Building JavaScript-based scalable software*. Indianapolis, Ind. : Wiley Wrox, 2013 (Wrox programmer to programmer). – ISBN 9781118240564

[VSW03] VAN DER AALST, Wil M. P. ; STOFFELE, Moniek ; WAMELINK, J. W. F.: Case handling in construction. In: *Automation in Construction* 12 (2003), Nr. 3, S. 303–320. http://dx.doi.org/10.1016/S0926-5805(02)00106-1. – DOI 10.1016/S0926–5805(02)00106–1. – ISSN 0926–5805

[Whi04] WHITE, Stephen A.: Introduction to BPMN. (2004)

[Wik08] WIKIMEDIA FOUNDATION: *MediaWiki version history*. https://en.wikipedia.org/wiki/MediaWiki_version_history. Version: 2008