Fakultät für Informatik

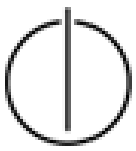
der Technischen Universität München

Bachelorarbeit in Wirtschaftsinformatik

Interactive Management of Attributes with Types, Formats, and Constraints in an Enterprise 2.0 Tool - Design and Implementation

Interaktives Management von Attributen mit Typen, Formate und Integritätsbedingungen in einem Enterprise 2.0 Werkzeug – Design und Implementierung

Author:	Matti Maier
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Christian Neubert
Submission Date:	August 11, 2011



I assure the single handed composition of this bachelor thesis only supported by the declared resources.

Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

_____

_____

Matti Maier

Abstract

The amount of data being stored and transferred every day is rising and with it the quantity of unstructured data. This data is difficult to evaluate and to use. Therefore a Hybrid Wiki system allows the user to specify attributes as key-value pairs describing the content of the wiki page and making it possible to query for similar or equal attributes. The value of an attribute can be typed to allow better comparison between attributes and to apply different formats, graphical representations and constraints.

In this thesis a list of types is collected by researching the literature and online resources. The list contains the name, formats, graphical representations, possible values and constraints for each type. Then software products, for example Microsoft's Excel spreadsheet software, the SAP ERP software suite and open source software such as OpenOffice and SugarCRM are examined for their use of typing, formats and representations. Using this information a technical solution is designed covering the examined aspects of types and finally implement the solution as part of the Tricia Enterprise 2.0 tool.

The objective of the work is to find a mechanism that allows the Hybrid Wiki user to display the values of an attribute in a format compliant with his or her locale, to support constraints for attribute values and to have an unified representation of all attributes with the same type.

Index

Motivation and Problem Statement

During the last fifteen years the Internet has become a crucial part of the world we live in. The amount of information being transferred and stored rises rapidly and with it the amount of unstructured data. Within enterprises unstructured data is stored in text files, documents, presentations, E-Mails, notes, protocols and in other files and information systems. Besides or as a document management system, enterprises have implemented wikis to manage their data. "A wiki is a web-based software that allows all viewers of a page to change the content by editing the page online in a browser. This makes the wiki a simple and easy-to-use platform for cooperative work on texts and hypertexts." [Ebersbach/Glaser/Heigl 2006, p. 12] Due to the restrictions within enterprises and by law, the access to view and edit pages in a wiki has to be limited. Furthermore the need for professional support as well as additional functionality such as the integration into an existing infrastructure require an extended wiki system often referred to as an enterprise wiki.

One of the key aspects of the next generation of the world wide web, also named Web 2.0, was the idea to give website visitors the ability to change and produce content. This concept was implemented using wikis. Enterprises had to adapt the Web 2.0. Those who did “have seized on collaboration and self-organization as powerful new levers to cut costs, innovate faster, cocreate with customers and partners, and generally do whatever it takes to usher their organizations into the twenty-first-century business environment.” [Tapscott/Williams 2006, p. 2] One tool to herald the Web 2.0 is an enterprise wiki. Within an enterprise wiki the information access is easier, for example by using a full-text search or browsing through a collection of pages, but the main problem persists: The most valuable information is in the prose text of a page and such a text cannot be queried or compared. To use this information an evolution of the enterprise wiki is required. A stores structured information with an open-templating mechanism and unstructured information in the body of the page. [Buckl/Matthes/Neubert/Schweda 2010, p. 6]

A allows the user to add attributes to a wiki page. An attribute consists of a key and one or more values. For example, a wiki page about a project can have an attribute with the key “project manager” and the value “John Doe”. Another attribute can be “project members” with the names of the project members as values or the key “project completion” with the completion date as a value. Since enterprises run many projects over time and each project wiki page can be associated with the same attributes, it is possible to define a template which is called a type tag. “Type tags allow the user to make a statement about the type of the object being described on the page.” [Matthes/Neubert/Steinhoff 2011, p. 3]. In the context of the project example above, the type tag can be called “project” and it can comprise attributes such as “project manager”, “project members”, “project status”, “project duration” and others. The specification of an attribute for a type tag is an attribute definition. On every wiki page with the type tag “project”, its attribute definitions will suggest the same attributes as other pages tagged with “project”. However, attribute suggestions are not merely based on type tags.

The benefits of this solution are the possible comparison of pages with the same type tags, the possibility to query the system for a set of pages and a quick overview of one wiki pages' contents. A full-text search relies on keywords and indices and is not capable to find all projects with a duration of more than six months, for example. To elevate the capabilities of comparing similar wiki pages, the value of an attribute can be typed. A type is defined either by adding new typed attributes to a page or by specifying the type in the attribute definition. In addition, a type can be defined to be strictly enforced in the context of an attribute definition. This enforcement is called a constraint and affects the comparability and usability of the system. If, for example, a value has the type *Date* on the instance level or defined by an attribute definition, it is possible to sort multiple date values in

an ascending or descending chronological order. Thus it advances the comparability of pages. Supplementary the value can be formatted according to the locale of the user. This improves the collaboration of people with different locales and therefore contributes to a more efficient team work. These goals can only be achieved if the added typing functionality does not result in a decrease of usability. Therefore the implementation has to focus on the usability of the graphical user interface (GUI) and the recognition of user input – or simply the interactive management.

To summarize the concepts, the following Figure depicts the relation of the terms defined above:

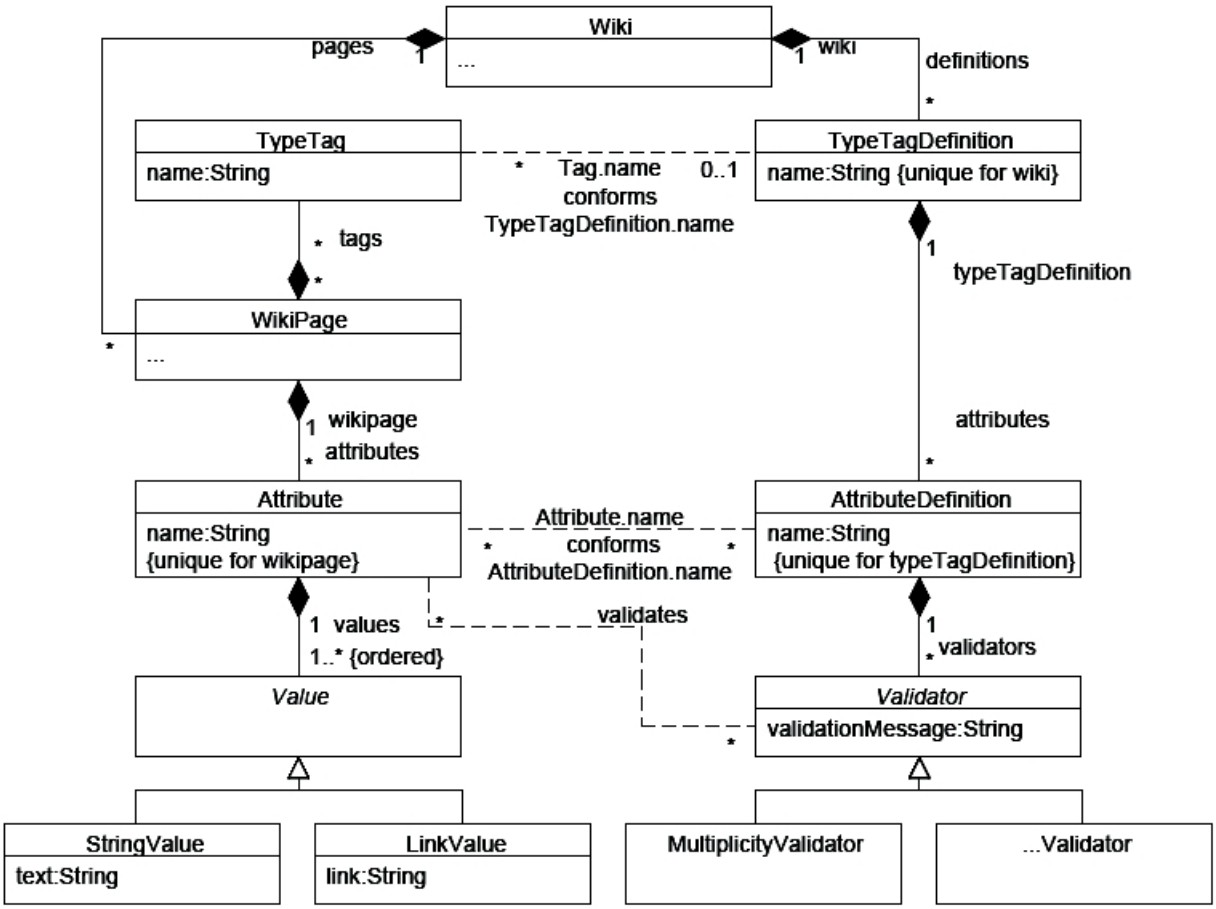


Figure 1: The Data model of Hybrid Wikis. [Matthes/Neubert/Steinhoff 2011, p. 7]

A *Wiki* contains none, one or more instances of a *WikiPage* and it comprises none, one or more instances of a *TypeTagDefinition*. A *TypeTagDefinition* has a unique name in the scope of a *Wiki* and defines the contents of a *TypeTag*. Technically a *TypeTagDefinition* and a *TypeTag* are coupled by their names. None, one or more *WikiPages* can comprise none, one or more *TypeTags*. These *TypeTags* influence the suggestion of attributes. A *WikiPage* can contain none, one or more *Attributes* while each of the attributes has a unique name in the scope of a *WikiPage* and an ordered list of typed values. *AttributeDefinition* and *Attribute* are coupled like *TypeTag* and *TypeTagDefinition* by their names. Practically the *TypeTagDefinition* contains *AttributeDefinitions* to constrain *Attributes* using *Validators*. A *Validator* can restrict the multiplicity of the attribute values, their types and the strength of enforcement. The strength of enforcement can either be strict in which case the attribute values have to be compliant to the validators, or the enforcement can be non-strict in which case a non-compliant value results in a warning *validationMessage*.

Outline

This paragraph outlines the completed work within the scope of this thesis. At first, data types are collected by researching the literature and observing software tools. For the latter, a choice of software tools which had already solved the problem in the past has to be made in order to evaluate how these tools handle types, formats, constraints and the graphical representation. As a result a detailed list with most commonly used types, with their formats, possible values and graphical representation examples, is created. With this list at hand the requirements for a preliminary design is determined and elaborated. Afterwards, the preliminary design is mapped to the Tricia platform in order to develop a detailed system design. During the validation process of the software design minor changes have to be applied.

In the beginning of the development phase the Tricia platform configuration is changed to save the configuration parameters that are necessary to initialize the system's language settings. Afterwards the core of this implementation has to be developed. Classes with their attributes and methods are outlined and tests are being written. After the tests are completed, the core classes are implemented and tested with the previously written tests. Furthermore the graphical user interface is modified to accommodate the new functionality and bugs have to be removed from the implementation.

Preparations

If a developer thinks of data types, the result will be types[1] like boolean, integer, floating-point numbers, characters and text. For an experienced Microsoft Excel user types such as number, currency and date are most likely to be mentioned. A librarian would most likely mention an ISBN number or a book category as a possible type. In order to get an overview of the most significant data types, a list had to be created. The following paragraphs describe the use of types, formats, constraints and graphical representations in five selected software tools.

Literature Research

Christian Ullenboom describes in his book *Java ist auch eine Insel* the primitive data types *boolean*, *char*, *byte*, *short*, *int*, *long*, *float*, *double* and *String* [Ullenboom 2009, p. 103-105]. In addition to the primitive types for numbers and strings, the type *date* is handled as such by database management systems. Furthermore these systems offer the types *binary large object* (*blob*) and *raw* for files and media content and lately the type *xml* was added to store extensible markup language (XML) files [Eickler/Kemper 2009, p. 110]. Angela Scheller has created a more detailed list with additional types such as *enumeration*, *sequence*, *empty/null type*, *identifier*, *reference*, *placeholder* and *entity type* [Scheller 1993, p. 99-100].

In detail, a primitively typed value can be converted into a sequence of bits and stored on a physical device. For example, the parts of a date, such as year, month, day, hour, minute, second and fraction of a second, are converted into numbers and these are again converted into bits and stored on a hard drive or in the memory. A famous example is the Unix timestamp which represents every date as the number of seconds since January 1st, 1970 0:00am Greenwich Middle Time (GMT). Then the numeric value is represented in the binary system and each digit interpreted as a bit in order to save the bits to a physical device. In contrast to primitive types, which characterize single values, complex types describe multiple values. A database Table, for example, has several columns each typed with a primitive type. If the database Table were one for cars with columns such as make, model, year of manufacturing, et cetera, then each row would have to be considered an instance of "car". The row itself would be of the complex car type. This thesis focuses on primitive types for attributes in a Hybrid Wiki.

From the user's point of view, the usability of the GUI is of crucial importance. „A user interface is simply the means by which an application communicates with the user, and the user with the application. The effectiveness and user acceptance of an application are determined primarily by the design of the user interface." [Markus/Smilonich/Thompson 1995, p. 3] Therefore it is necessary to research the basics for successful interactive management. The user interface environment can be divided into three major aspects: human factors, presentation and interaction.

Human factors are about the human's response to the application. The first principle is to "allow the user to control the type of information presented. The more the user feels in control, the more the user will be comforTable and satisfied with the application." [Markus/Smilonich/Thompson 1995, p. 4] Furthermore the user should be able to customize the interface such as s/he is able to change interface attributes such as color, function, structure and content. Additionally the navigation to functions should not require the user to memorize a set of complex commands or options. [Markus/Smilonich/Thompson 1995, p. 4]

1 Please note that the terms "data type" and "type" are used equally in the following paragraphs.

Another factor is the application's presentation. An aesthetic appeal as well as meaningful and recognizable representations are as important as a consistent interface. „A consistent interface refers to the similarity in appearance and layout of the components. A more critical aspect of a consistent interface is functional consistency. Functional consistency means that the same action should have the same result regardless of the mode the application is in.“ [Markus/Smilonich/Thompson 1995, p. 9]

As a third factor, the interaction between the human and the application is in focus. It is necessary to provide immediate feedback to the user's manipulations on the system. This increases the user's rate of learning and it gives the user a better perception of the application. Still the system should accept “user actions that do not conform precisely to system specifications without negative consequences to the user.“ [Markus/Smilonich/Thompson 1995, p. 12] This can be achieved by implementing an *Undo* function, confirmation dialogs for destructive actions and disabled options on the screen.

To summarize the findings in the literature, the design and implementation phase has to consider various data types and it has to create a meaningful, aesthetic and consistent graphical user interface. In regard to the human factors, the interaction between the application and the user needs to offer immediate feedback and the ability to redo changes.

Evaluation of Software Tools

Most of the types described in the literature are used to develop a software system. Nevertheless these types are often not visible to the user of a software tool. Therefore it was necessary to evaluate applications with a typing system and a wide range of users as an indicator for its acceptance.

Microsoft Excel 2010

At first, Microsoft's spreadsheet application ***Excel 2010*** was examined. The software tool is utilized by a variety of different users, for example students, projects managers, financial analysts and engineers, who mainly use the software for tabular, structured data. The data provided in cells can be typed, formatted and used for calculations in formulas. In general Excel only supports only three basic data types: numeric values, text and formulas [Walkenbach 2010, p. 29]. Still Excel offers various formats for a cell: *General*, *Number*, *Currency*, *Accounting*, *Date*, *Time*, *Percentage*, *Fraction*, *Scientific*, *Text*, *Special* and *Custom*. The format *General* is a placeholder for the tangible format of the cell's content. By default it is selected and the content is interpreted after the user enters the value and confirms it. The value is displayed according to Excel's interpretation. In contrast to the types depicted in the literature above, Excel does not distinguish between an integer and a floating-point number, all numbers are numeric values and formatted as a number, fraction or scientifically. Accounting is a format where the amount of decimal places can be specified along with the currency whereas the format Currency can additionally specify the color of a value, for example, if the number is negative the number is presented in red.

If the content of a cell is interpreted as a date or the format *Date* is applied to the cell, the value is converted internally to a floating-point number. Dates before January 1st, 1900 are not converted, but saved as text while every date after January 1st, 1900 is the amount of days since then. The time is converted into a fraction of a day.[2] For example, 6:00am is 0.25. If January 2nd, 1900 at 6:00am is given, Excel converts this date to 2.25.

2 Verwendung von Datums- und Zeitangaben in Microsoft Excel, http://support.microsoft.com/kb/214094, Accessed: July 5th, 2011 4:00pm.

A numeric value can be formatted using the *Percentage* format and thus it is presented with a percentage symbol and two decimal places by default. If the format *Text* is selected for a cell, the content is interpreted as a sequence of alphanumeric and symbolic characters. The format *Special* can be applied to mark the content of a cell as a zip code, social security number or ISBN number. The *Special* formats vary with the version of Microsoft Excel 2010. The German version has different settings for the zip code, than, for example, the US version. With the format *Custom* an user can specify an individual format for the cell's content by using a defined set of codes [Walkenbach 2010, p. 557-558].

Furthermore the value of a cell can be a reference to another cell or contain a calculated value. For these purposes the cell is typed as a *Formula*. It does not specify the format of a cell's value, it only calculates the cell's value. In addition to the types applicable for single cells, a "worksheet can also hold charts, diagrams, pictures, buttons, and other objects. These objects aren't contained in cells. Rather, they reside on the worksheet's *draw layer*, which is an invisible layer on top of each worksheet." [Walkenbach 2010, p. 29]

As the next step in line was to examine how values are stored, the documentation of Microsoft Excel 2010[3] provided a profound basis. "All worksheet numbers in Excel are stored as doubles so that it is not necessary (and in fact introduces a small conversion overhead) to declare add-in functions as exchanging integer types with Excel."[4] Together with the rational stated by John Walkenbach that only three data types are utilized by Excel – namely numeric values, text and formulas – and the fact that the formats on a cell differ in the specific version of Excel for different locales the following mechanism can be concluded:

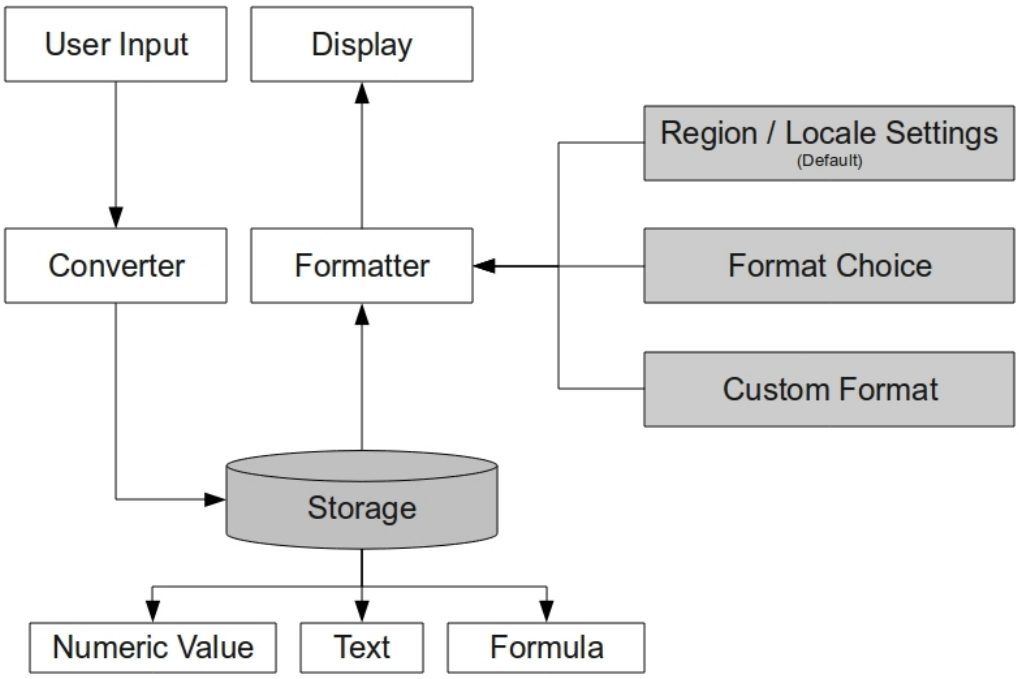


Figure 2: Microsoft Excel Storage Mechanism [Source: Author's design]

3 MSDN Library: Excel 2010, http://msdn.microsoft.com/en-us/library/ee658205.aspx, Accessed: July 6th, 2011 11:00am.

4 MSDN Library: Data Types Used by Excel, http://msdn.microsoft.com/en-us/library/bb687869.aspx, Accessed: July 6th, 2011 12:30pm.

As depicted in Figure 2, the user enters a value into a cell (*User Input*). The value is converted (*Converter*) to either a number (*Numeric Value*), *Text* or *Formula* and saved in the *Storage*. In order to display the value on the user's screen, it is retrieved from the storage, formatted with either a previously defined custom format for the cell or the defaults for the internationalized version of Excel, and finally written onto the *Display*. With this level of knowledge it was possible to examine the graphical representations of the value. Therefore the output can be divided into three different categories: formatted value, error and formula. If Excel was able to interpret the user's input with the preset format, it would confirm the input and consequently show it in the cell. If the extent of the cell was not enough, an error message would have to be displayed by simply filling the cell with hash-symbols (#), otherwise the formatted value would be displayed. In case the value was a valid formula and the cell's extent was adequate, the result of the calculation is displayed. Only when the cell is edited, the formula is shown in the cell and in the designated field. If the formula was not valid, another error message "#NAME?" is shown in the cell and a symbol next to the cell tries to give advice on how to solve the problem. Excel has further ways to present error messages such as dialogs, warnings, error symbols and other error types.[5]

Constraints can be applied in Excel with formulas or the specification of a format for a cell. For example, if the cell is required to contain dates, the format *Date* can be applied to the cell so that every user input will be regarded as a date. In addition, Excel offers a function to determine the data type of a cell[6] and this can be used by a formula to check on the cell and take action if the type of a cell changes to an invalid value. Other constraints such as multiplicity constraints can be handled by other formulas and functions.

Microsoft Access 2010

Another tool within Microsoft's Office software suite is *Access*. Similar to the use of Excel, a variety of users utilize Access for structured data, but often with the need to query specific data. Since structured queries are also possible with a Hybrid Wiki, the evaluation of Microsoft Access in addition to Excel was essential. The software manages relational databases and offers functions to edit, view, remove, filter and define the data. During the process of defining the data scheme, one of the nineteen data types has to be specified for each column of a Table.[7] Thus all values are typed and the types are automatically enforced. Other constraints, for example if no value is allowed, have to be specified in the data scheme and are also strictly enforced within the scope of the definition. In addition to the previously mentioned types, a new type called *AutoValue* was introduced in Access 2007. [Kolberg 2007, p. 97-98]. This type is a placeholder for automatically calculated values often used for identifiers. In the release notes for Access 2010 further type incompatibilities between Access and Microsoft's SQL Server 2008 are depicted.[8]

The following Figure shows the data sources Microsoft Access 2010 is connectible with and further depicts the field types available in the software:

5 Microsoft Office Support: ERROR.TYPE, http://office.microsoft.com/en-us/excel-help/error-type-HP005209079.aspx, Accessed: July 6th, 2011 1:00pm.

6 MSDN Library: Retrieving the Values of Cells in Excel 2010 Workbooks by Using the Open XML SDK 2.0, http://msdn.microsoft.com/en-us/library/ff921204.aspx, Accessed: July 6th, 2011 3:00pm.

7 MSDN Library: Microsoft Access Data Types, http://msdn.microsoft.com/en-us/library/ms714540(v=vs.85).aspx, Accessed: July 6th, 2011 3:30pm.

8 Microsoft TechNet: Changes in Access 2010, http://technet.microsoft.com/en-us/library/cc179181.aspx, Accessed: July 6th, 2011 4:30pm.

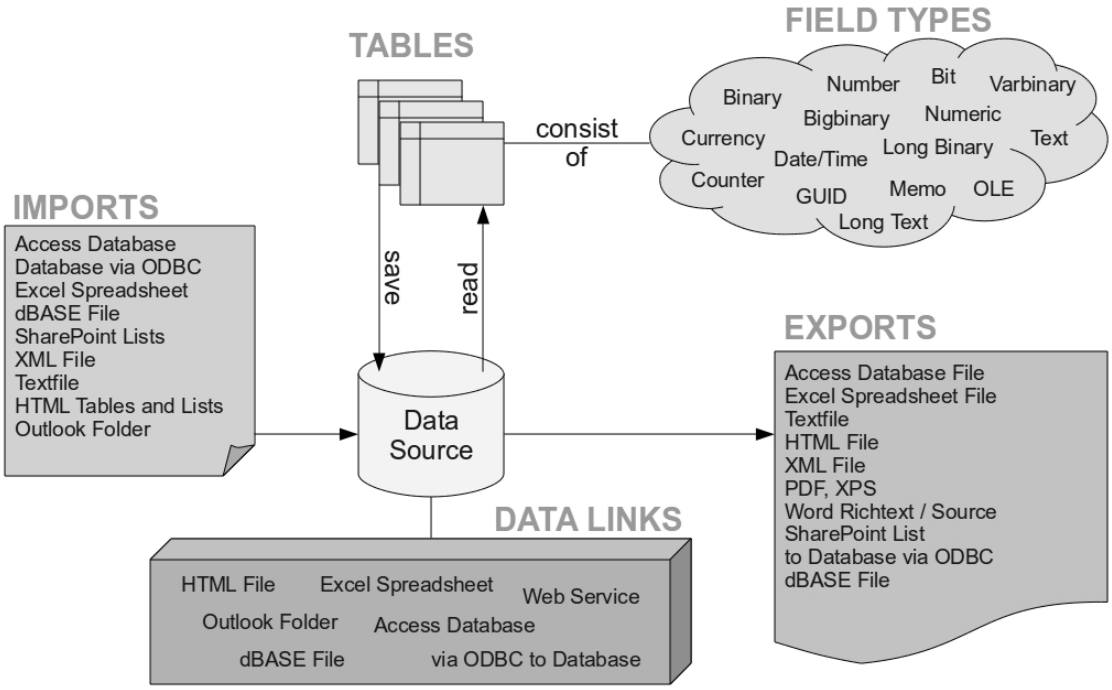


Figure 3: Connections and Field Types in Microsoft Access 2010
[Source: Author's design]

The structured data stored in Microsoft Access 2010 is stored in Tables. Each Table column has one of the field types shown in Figure 3.[7] The field types in the Figure correspond to the field types the user is able to choose from in the graphical data definition tool in Access. A data link embodies the connection to a data source shown in the DATA LINKS box in Figure 3. On saving a Table, it is stored using a data link. Access is also capable of importing data from other data sources such as the ones listed in the IMPORTS box in Figure 3. The export of databases is supported to containers specified in the EXPORTS box. The user manages the data sources as well as the other functions provided by the software via the Access GUI. There s/he also edits the data in a manner similar to Excel, but with a previously defined, fixed amount of typed columns. If the input value does not correspond to the type selected for the column the user is confronted with the choice to either enter a new value or change the type of the column. Since Access does not support multiple values per cell, a reference to another Table can be created.

Another aspect under evaluation were structured queries. Access 2010 offers multiple ways to filter a Table, for example, to filter rows with a certain value or a range of values in one or more columns. Filters can be created using three different graphical tools. The first simply applies a filter to one column, the second tool offers to build a form based filter for multiple values. The third tool shows a Table where detailed information about the priority of more than one filter and the result order can be specified. Furthermore Access provides options to sort the Table rows in a ascending or descending order. Values are sorted type dependent which enables Access to sort formatted values correctly, not just by applying a simple lexicographic sort algorithm. More complex queries can be formulated in SQL (structured query language). One can either write the SQL queries and send them to Access or s/he can utilize one of the four graphical assistants to build the required query.

Although both Excel and Access focus on editing structured data, the way the user deals with the data is very different. The tools have in common that they are both able to work with typed values, format values and restrict values to be of a certain type. In detail, the available types differ, for example, Excel does not offer a binary data type and Access can deal with dates before January 1st,

1900. In Excel types are specified with their formats while Access explicitly defines the types in a data scheme. The type in Access applies to the whole column while in Excel one column can have multiple types. In addition, cells in Excel do not need to be typed while cells in Access are always typed. This circumstance also affects the strict enforcement of types in Access and the rather tolerant enforcement in Excel. Finally the way how both tools' graphical user interfaces define types, specify formats and apply constraints, vary widely. While both tools offer to edit structured data similarly, the interfaces to view and edit data differ. While Excel does not offer to filter a Table without using complex formulas, Access provides multiple ways to filter and sort Tables. Still both tools are able to reference data outside the scope of the Table. Excel uses formulas while Access uses a foreign key for references. The evaluation of both software tools has shown different opportunities to deal with structured data.

OpenOffice.org

In addition to the different possibilities offered by Microsoft's Office products to input, store, format and display data, the open source software package OpenOffice.org 3.1.0 was evaluated. The access to the source code as well as the detailed documentation presented a profound basis for the analysis of the typing system used in OpenOffice.org. One of the results found, was that a date value is stored as a floating-point number like in Excel, but the formatting is handled by a *NumberFormats* service which implements two interfaces *XNumberFormats* and *XNumberFormatTypes*. The first one provides access to the number formats of a container and the latter contains predefined formats such as percentage, time, currency and text. A container can be a page, a Table, the document itself or a Section of a document. The general idea is to store formats in each container and add custom formats or predefined formats to a container when they are being used. All the formats are accessible with a unique index key that can differ from container to container. [OpenOffice.org 2009, p. 814-819]

SAP ERP

As this thesis aims at enterprise software, the SAP ERP software suite was evaluated as one of the largest representatives of enterprise resource planning (ERP) tools with over one million users. Since version R/3 in 1992, the product is based on relational databases and it offers a consistent user interface. [Hansen/Neumann 2005, p. 544] The supported types are equal to the SQL Standard from 1992[9], but extended with the types available in the SAP database specification[10]. During the customization process of the SAP ERP product, formats for dates and times can be specified.[11] This process is less user friendly than in Microsoft Office or OpenOffice.org, because the user has to remember the correct codes. The fields on the graphical user interface strictly enforce the use of the correct type and always show the value in the correct format. This behavior is similar to Microsoft Access.

9 Information Technology - Database Language SQL, http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt, Accessed: July 7th, 2011 11:00am.

10 SAP-Bibliothek: Spezifikation von Werten (value_spec), http://help.sap.com/saphelp_fica472/helpdata/de/48/0d8018b4f211d2a97100a0c9449261/frameset.htm, Accessed: July 7th, 2011 11:15am.

11 SAP-Bibliothek: Datums- und Zeitformat (datetimeformat), http://help.sap.com/saphelp_fica472/helpdata/de/48/0d8018b4f211d2a97100a0c9449261/frameset.htm, Accessed: July 7th, 2011 11:15am.

SugarCRM

Lastly an enterprise application in the context of the Web 2.0 was evaluated: the Community Edition of SugarCRM 5.2.0. The website[12] describes the software as a "Open Source Business & Social CRM Software" that "helps your business communicate with prospects, share sales information, close deals and keep customers happy." The source code of SugarCRM is available under the GNU Affero General Public License in Version 3[13] and therefore it is visible to the public.

In the user interface values are not explicitly typed, although some fields require to supply a value of a certain type such as a date, time or numeric value. Some of the input fields are only checkboxes, radiobuttons or selection menus which only allow the user to choose from a defined set of values. Thus the types used by SugarCRM are dates, times, numbers, text, boolean and enumeration. The constraints enforcing these types are programmatically set by the application developers. Each time a typed value is displayed, the system automatically formats it according to the format specified in the user preferences. If the user has not adjusted the settings, the system settings are used instead. By default the formats for all types are specified in an array which is included in the application's data and installed during the initialization process. Similar to the SAP ERP system, SugarCRM stores data in a relational database. Newly supplied values are formatted for the use in an SQL query and stored in the database. If the value is requested again, it is retrieved from the database, formatted by a class which represents the type, for example date and time values are represented by the class *TimeDate*, and appended to the response.

A storage mechanism similar to the one depicted in the previous Section is common to all the tools evaluated. An user supplies a value which is evaluated and prepared to be stored in a database or something alike. In order to view the value, it is reformatted and written onto the screen. Although the types vary widely, some types such as date, number and text are used by all of the systems. Furthermore types such as boolean, currency, percentage and enumeration can also be handled by more than two software tools. Although all software systems share a set of types, their graphical representations differ. Microsoft Excel 2010 only allows to specify a type by providing a format for a value in a separate dialog. Microsoft Access 2010 does not allow to specify a type for a single cell, only for a whole column and data definitions are made in a separate editor within the tool. OpenOffice.org handles types and formats within containers, but the dialogs within the spreadsheet application are similar to the one displayed in Microsoft Excel. In contrast to the desktop applications SAP ERP types and formats every value, but defines the value's type during the application development process. Likewise SugarCRM defines a value's type during application development and also enforces the type's usage in a strict manner. While OpenOffice.org Spreadsheet and Excel do not enforce the type constraints as strict as Access, the types can only be changed programmatically in SAP ERP and SugarCRM.

12 SugarCRM Website: http://www.sugarcrm.com/crm/, Accessed: July 7th, 2011 12:00pm.
13 GNU Affero General Public License: http://www.gnu.org/licenses/agpl.html, Accessed: July 11th, 2011 1:00pm.

List of Data Types

As a result of the above findings, the following list shows the most significant types:

Name of the Type	Example Representations
Binary	image, audio, video data
Boolean	0/1, true/false
Currency	123€, 123.45 EUR
Date	January 1st, 1970 10:01am GMT
Enumeration	[A, B, C, D]
Number	123, 123.4, .25, 0.123
Percentage	12%, 1.23 %
Reference	identification number or string
Text	plain text, formatted text

Table 1: List of Common Data Types

The example representations depicted in Table 1 are text representation examples for each of the types. Graphical representations as well as possible values, formats, order, constraints and saving options can be found in a compilation of types in Appendix A.

Software Design

Building on the initial research and evaluation phase, the following paragraphs focus on the requirements and the software design. At first, the initial situation is described followed by the challenges being addressed. Afterwards requirements are determined and use cases are defined. On the basis of these insights a gross design is specified and developed to a platform-related system design. Finally the sketch for the graphical user interface will be outlined.

The Initial Situation

An implementation of a Hybrid Wiki is the Tricia platform. The platform's server component is based on Java while the client which runs in a browser is based on Hypertext Markup Language (HTML), Cascading Style Sheet (CSS) and Javascript with the use of jQuery[14] and jQuery-UI[15] as well as other jQuery plug-ins such as DataTables[16]. The data stored in Tricia is saved to the server's file system, a database, Extensible Markup Language (XML) configuration files, text indices and templates. Alongside with the Hybrid Wiki features offered by the platform, it also includes other functionality, for example file and directory sharing, site management, blogging and social networking.[17]
Currently the Hybrid Wiki component of the platform supports to define attributes for each wiki page and attribute definitions based on type tags. An attribute value can be typed as a text value or link value. Values typed as text represent all values that do not refer to another internal or external resource. An internal link is a reference to an entity such as a page or an user within the system while external links refer to resources outside the system such as other websites. Link values are formatted so the user can navigate to the resource by clicking on the link. In general, text values are not formated except for the values “true” and “false” which are displayed as a checkbox that is checked for “true” and not checked for “false”. Attribute definitions can define a type such as *Date* or *Boolean* and decide whether to strictly enforce the use of these types, but internally the values are still typed as text. Such value restrictions are also referred to as constraints. Furthermore the system provides a tabular overview for pages tagged with the same type tag. The Table can be lexicographically sorted by single columns and the attribute values can be edited by clicking on the cell's edit icon. The platform also provides basic internationalization features such as the assignment translation of messages.

For better comparability of attribute values, for example within the tabular overview of pages with the same type tag, more types are necessary. Structured queries directed at the system will be able to decide whether a value should be selected for the result set based on internal value comparison. Additionally, values typed as number or date can be formatted considering the user preferences. Thus the language setting in the user preferences need to accommodate settings for date and time separation symbols, the order of the date, for example whether it should be ordered in an American fashion as month/day/year or in a European format like day.month.year, as well as the decimal and thousand separator. This increases the usability, readability and productivity, because an user is able to interpret the given dates and numbers faster and misunderstandings between employees are reduced. The newly introduced types must also be integrated in the abilities of attribute definitions.

14 jQuery Project, http://jquery.org, Accessed: July 12th, 2011 10:15am.
15 jQuery UI – Home, http://jqueryui.com, Accessed: July 12th, 2011 10:15am.
16 DataTables (Table plug-in for jQuery), http://www.dataTables.net, Accessed: July 12th, 2011 10:15am.
17 infoAsset : Explore Tricia, http://infoasset.de/wikis/infoasset/tricia, Accessed: July 12th, 2011 10:30am.

Requirements

As a basis for the system design the requirements for the project need to be defined. In the paragraph above it is explained why more types are necessary for the platform in order to increase its usability and functionality. During the system design phase it is essential to keep the usability high while adding more functionality. Besides this fundamental requirement, the following types have to be added to the platform: date/time, number, percentage and currency. The other types depicted in the list of data types above are already included in the system, except for the binary type. In addition, every typed value should be formatted according to the user's locale which is specified in his or her account settings. The user interface should implement a feedback mechanism to respond to incorrectly typed values if the type is set by an attribute definition. Visual objects such as the date icon should be hidden if no date or time is entered. After the user has completed his or her input, the value needs to be evaluated and an immediate result has to be visible. If the result is not in the user's intent, s/he should be able to change the type. If the input value is not compliant with the type, a warning has to be issued, but the value should not be removed.

Furthermore the configuration of Tricia has to be changed to accommodate the available locales with their parameters. Every locale is identified by a key which consists of the country code and language as specified in the ISO 3166-1 and ISO 639-1. Additionally every locale is named for better readability. An administrator has to be able to set the default locale for the system and groups. If an user does not choose his or her locale, the default locale replaces the choice. Other functions to edit and remove locales need to be implemented in the configuration system, too.

Use Cases

On the basis of Figure 4, the following paragraphs describe the use cases for which the solution has to be implemented. The use cases shown are only a relevant subset of all use cases for the Hybrid Wiki system. To start from the premise that all Hybrid Wiki users can be assigned to either the administrative or the user role, all use cases except for two can be associated with the employee who is not responsible for application and server maintenance. Users can be either employees, externals or any other stakeholders within an enterprise while an administrator is designated to make sure the users are able to access the software.

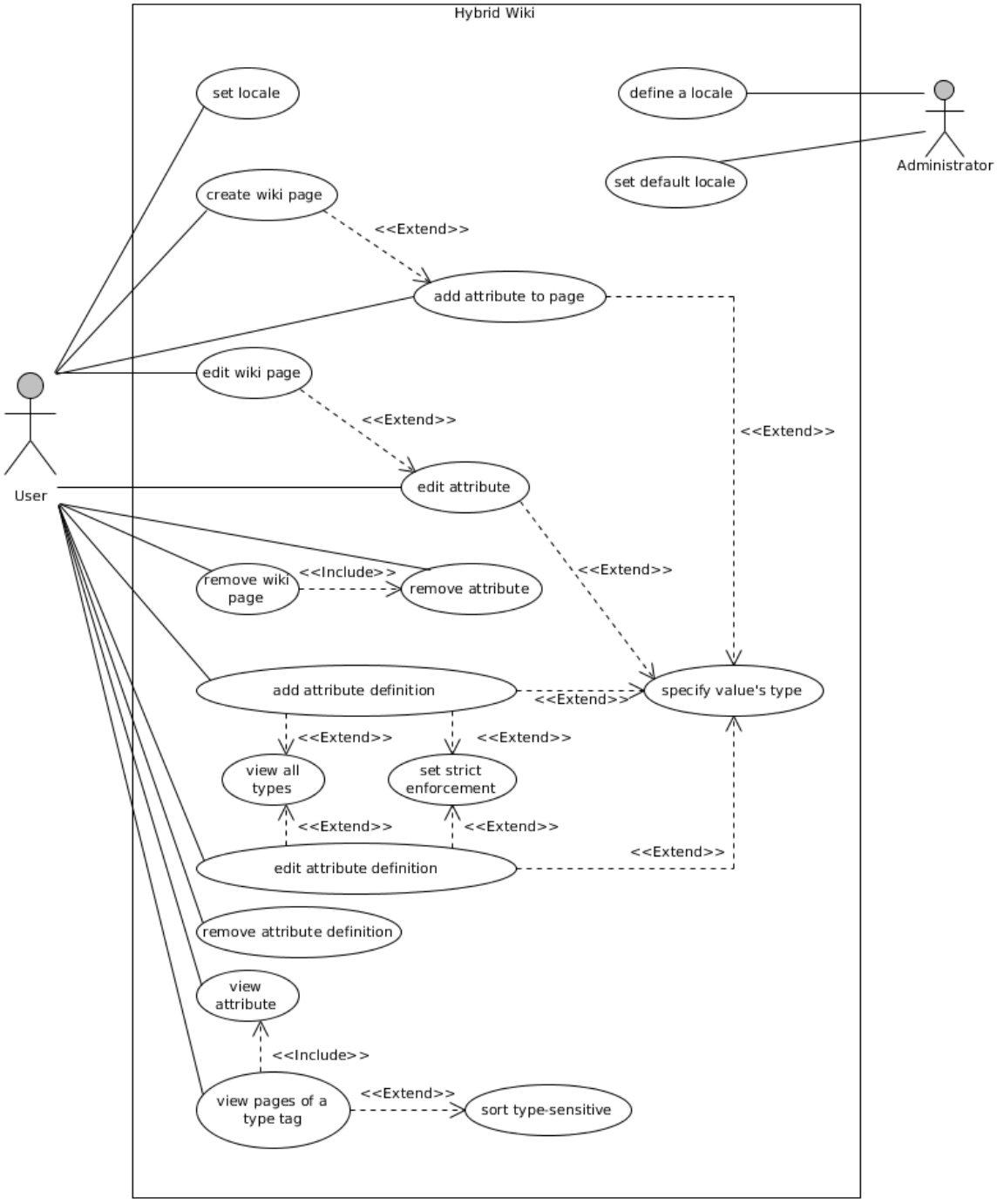


Figure 4: Use Cases for a Hybrid Wiki [Source: Author's design]

Set Locale

The user can set his or her locale for the user account. Normally s/he opens the account preferences and sees an option to choose the locale. If the user clicks on it, all choices are shown and s/he can decide which one is preferred. Finally the changes are saved and implicitly all values are formatted according to the chosen locale.

Use Case Name	**User Settings: Set Locale**
Actors	User, Administrator
Trigger	User-Account Configuration
Preconditions	One or more locales are available.
Postconditions	None or one locale is set.
Invariant	Other users' settings are not affected.
Normal flow	1. Open account preferences 2. Show all locale choices 3. Save choice
Alternative flow	1. Open account preferences 2. Show default settings 3. Click on default setting to change it 4. Save changes

Table 2: Set Locale

Set Default Locale

The administrator sets the default locale for a group or all users. This locale is applied if an user has not selected a locale. In order to choose the default locale, the administrator opens the system configuration, selects the locale of his choice and saves the setting.

Use Case Name	**System Settings: Set Default Locale**
Actors	Administrator
Trigger	System Configuration
Preconditions	One or more locales are available.
Postconditions	One locale is set.
Invariant	The locales are not affected.
Flow Group	1. Open group configuration 2. Select locale 3. Define it as default
Flow System Default	1. Open system configuration 2. Find *Default Locale* setting 3. Choose the default setting

Table 3: System Settings: Set Default Locale

Define a Locale

Normally when a new instance of the system is installed, the locales are being defined. Therefore the System configuration is opened, the dialog to set the id and the name of the locale is requested and the parameters are specified. Finally the locale is saved and ready to be applied.

Use Case Name	System Configuration: Define a Locale
Actors	Administrator
Trigger	System Configuration
Preconditions	System configuration is accessible.
Postconditions	Locale is saved.
Invariant	Other locales are not affected.
Normal flow	1. Open system configuration 2. Request to add a new locale 3. Enter name and identification of the locale 4. Define parameters 5. Save the locale

Table 4: System Configuration: Define A Locale

Create Wiki Page

When the user requests to create a new wiki page, the new page can be created with or without adding attributes. Page contents can be a description of the page, type tags or values for other available parameters. If type tags were specified the attribute definitions associated with the type tag have to be shown as well as the their types have to be visible for the user.

Use Case Name	Create Wiki Page
Actors	User
Trigger	-
Preconditions	The system is running and the user has the privilege to add a new wiki page.
Postconditions	The new wiki page is saved.
Invariant	Other wiki pages are not affected.
Normal flow	1. Open dialog for new page 2. Specify name of the page 3. Enter contents 4. Save Page
Alternative flow	1. Open dialog for new page 2. Specify name of the page 3. Enter contents 4. Add attributes 5. Save Page

Table 5: Create Wiki Page

Specify Value's Type

Every time an attribute or attribute definition is added or edited, the value's type has to be specified. By default the type is set to *Text* and changes when the user input is recognized as another type.

Use Case Name	Specify Value's Type
Actors	User
Trigger	Attribute or attribute definition is added or edited.
Preconditions	At least one type is available.
Postconditions	One type is specified.
Invariant	The user input is not influenced by the type recognition.
Normal flow	Attribute added/edited 1. User enters value 2. Value is recognized as of a certain type and type is displayed 3. Save typed value Attribute Definition added/edited 1. Type is defined 2. Type is saved with the attribute definition
Alternative flow 1	Attribute added/edited 1. User enters value 2. Value is not recognized and thus the type is set to *text* 3. Save value with type *text* Attribute Definition added/edited 1. Type is not defined 2. Attribute definition is saved without a type
Alternative flow 2	Attribute added/edited 1. User enters value 2. Value is recognized as of a certain type and type is displayed 3. User changes the type 4. Value is saved with the user's choice of type

Table 6: Specify Value's Type

Add Attribute to Page

In order to add an attribute to a page, the user has to have the privilege to access and edit the page. The user can get to the attribute's dialog either from the page where s/he wants to add the attribute or from the editing perspective of the page or from a tabular view of type tags associated with the page. Once the user input has been recognized by the system and submitted the result to the user's view, the user can decide to change the recognized type. If the user tries to save the attribute, the system checks whether an attribute definition corresponding to the attribute's name is available and if the type specified in the attribute definition fits to the recognized or chosen type. In case no attribute definition is available or the attribute's type fits to the attribute definition, the value is saved. Otherwise a warning is issued so the user can change the type and value or abort the process.

Use Case Name	Add Attribute to Page
Actors	User
Trigger	-
Preconditions	The user has read and write privileges for the page and the system is running.
Postconditions	The new attribute is associated with the wiki page and saved.
Invariant	Other attributes and contents of the page are not affected.
Normal flow	1. Open attribute dialog 2. Enter attribute's name 3. Enter attribute's value 4. Save attribute
Alternative flow	1. Open attribute dialog 2. Enter attribute's name 3. Enter attribute's value 4. Change recognized type 5. Save attribute if no attribute definition applies to the attribute or if the type does not conflict to the attribute definition, otherwise a warning should be issued.

Table 7: Add Attribute to Page

Edit Wiki Page

Users can edit a wiki page either by selecting the area which needs to be changed or they can choose to open a screen to edit the current page. This use case includes to edit an attribute as described in the use case “Edit Attribute” below.

Use Case Name	Edit Wiki Page
Actors	User
Trigger	-
Preconditions	The user has read and write privileges for the page and the system is running.
Postconditions	The changes are saved and visible on the page.
Invariant	Other wiki pages are not affected by the change.
Normal flow	1. Select area to edit 2. Click on edit icon 3. Change the contents 4. Save the changes
Alternative flow	1. Open *edit page* screen 2. Edit contents 3. Save wiki page

Table 8: Edit Wiki Page

Edit Attribute

To edit an attribute the user can either click on the edit icon beside the attribute or open the edit wiki page screen. In both cases the attributes' names, values and types are displayed and can be changed. If the value of an attribute is changed, the new value will be evaluated for its type and eventually changed. If the user agrees to the new type s/he can save the attribute or page, otherwise the user is able to change the attributes type. Please refer to the use case “Specify Value's Type” for details on the latter scenario.

Use Case Name	Edit Attribute
Actors	User
Trigger	Can be triggered if the page is edited (see use case “Edit Wiki Page”)
Preconditions	Attribute exists for the page and can be edited
Postconditions	Changes are applied to the attribute
Invariant	Other attributes are not affected.
Normal flow	1. Open attribute dialog by clicking on the edit icon 2. Change name and values 3. Save attribute
Alternative flow	1. Open *edit page* screen 2. Change attributes 3. Change recognized type of the value 4. Save wiki page

Table 9: Edit Attribute

Remove Wiki Page

In case a wiki page has to be deleted, all attributes associated with the page have to be removed, too. If the user selects the field for removal, a confirmation request is displayed and the page and its attributes are removed upon approval.

Use Case Name	Remove Wiki Page
Actors	User
Trigger	-
Preconditions	The page exists, the user's privileges allow him or her to remove the page and the system is running.
Postconditions	The page and all its attributes are removed.
Invariant	Other wiki pages are not affected.
Normal flow	1. Request to delete the page 2. Confirm deletion process

Table 10: Remove Wiki Page

Remove Attribute

To remove an attribute from a wiki page, the user has to open the dialog to edit the attribute and select *delete*. The attribute is removed from the wiki page if the confirmation request is accepted.

Use Case Name	Remove Attribute
Actors	User
Trigger	The page is removed or the user has requested to remove the attribute.
Preconditions	The attribute exists, the user has the privileges to edit the page and the system is running.
Postconditions	The attribute has been removed from the wiki page.
Invariant	Other attributes and wiki pages are not affected.
Normal flow	1. Open attribute dialog by clicking on the edit icon 2. Click on *delete* 3. Confirm removal

Table 11: Remove Attribute

View All Types

In the event an attribute definition is added or edited, all available types have to be shown in order to choose the preferred type.

Use Case Name	View All Types
Actors	User
Trigger	Add or edit attribute definition
Preconditions	One or more types are available.
Postconditions	Types have not been changed.
Invariant	The list of types does not change.
Normal flow	1. Open attribute definition dialog 2. Open selection menu 3. Show all types

Table 12: View All Types

Set Strict Enforcement

The *isStrict* parameter of an attribute definition determines whether the multiplicity and type of an attribute definition will be enforced strictly or not. In case the attribute definition is set to be strictly enforced every attribute with this attribute definition has to match the attributes parameters. Whether or not the attribute definition is enforced strictly on the attributes can be selected with a checkbox.

Use Case Name	Set Strict Enforcement
Actors	User
Trigger	Add or edit attribute definition
Preconditions	Either the multiplicity or the type of an attribute definition is set.
Postconditions	• Attribute definition is saved with the given strength of enforcement. • Attributes which refer to the attribute definition cannot be added or edited if the parameters do not correspond to the attribute definition.
Invariant	Other attribute definitions are not affected.
Normal flow	1. Open attribute definition dialog 2. Check the *isStrict* checkbox 3. Save the attribute definition
Alternative flow	1. Open attribute definition dialog 2. Uncheck the *isStrict* checkbox 3. Save the attribute definition

Table 13: Set Strict Enforcement

Add Attribute Definition

Since attribute definitions enable the user to enforce a type and multiplicity to an attribute, the name of the attribute definition and implicitly the targeted attribute have to be supplied. Furthermore the option whether to enforce the constraints strictly has to be set. Related use cases are “View All Types”, “Set Strict Enforcement” and “Specify Value's Type”.

Use Case Name	Add Attribute Definition
Actors	User
Trigger	-
Preconditions	Type tag is selected and system is running
Postconditions	Attribute definition saved and associated with type tag
Invariant	Other attribute definitions and type tag are not affected.
Normal flow	1. Select the attribute definition icon 2. Supply name 3. Select type, multiplicity 4. Set is strict parameter 5. Save attribute definition
Alternative flow	1. Select *New Attribute Definition* 2. Supply name 3. Select type, multiplicity 4. Set is strict parameter 5. Save attribute definition

Table 14: Add Attribute Definition

Edit Attribute Definition

Similar to the “Add Attribute Definition” use case this use case refers an attribute definition and is also related to other use cases such as “View All Types”, “Set Strict Enforcement” and “Specify Value's Type”.

Use Case Name	Edit Attribute Definition
Actors	User
Trigger	Edit icon of an attribute definition clicked
Preconditions	Attribute definition and type tag exists and system is running.
Postconditions	Attribute definition is saved and associated with the type tag.
Invariant	No other attribute definitions or type tags are affected.
Normal flow	1. Click on edit icon for an attribute definition 2. Change name and parameters such as type, multiplicity and enforcement strength (*isStrict*) 3. Save attribute definition

Table 15: Edit Attribute Definition

Remove Attribute Definition

If an attribute definition is removed, the attributes previously associated with the attribute definition will not be removed. Still the attribute definition is removed from the type tag's set of attribute definitions.

Use Case Name	Remove Attribute Definition
Actors	User
Trigger	Selected delete option in *edit attribute definition* dialog
Preconditions	Attribute definition exists and system is running.
Postconditions	Attribute definition is removed, but attributes have not been affected.
Invariant	Other attribute definitions, attributes and type tags are not affected.
Normal flow	1. Click on edit icon for an attribute definition 2. Select delete 3. Confirm removal

Table 16: Remove Attribute Definition

View Attribute

Attributes are shown either on a page or in a Table where all pages associated with a type tag are presented. Typed attributes are formatted according to the user's locale and attributes with an attribute definition attached are marked as such.

Use Case Name	View Attribute
Actors	User
Trigger	A page or an overview of the attributes associated with a type tag is requested.
Preconditions	The attribute exists and is associated with a wiki page and the system is running.
Postconditions	Attribute, page and type tag have not been changed.
Invariant	Attribute, page and type tag are not changed.
Normal flow	1. Request a page 2. View the attributes
Alternative flow	1. Request to see all pages associated with a type tag 2. View the attribute in a Table

Table 17: View Attribute

View Pages of a Type Tag

All pages associated with a type tag can be viewed by clicking on a type tag. The first column is the page's name followed by all other attributes associated with the pages and the attribute definitions for the selected type tag. The cells show the attributes' values, thus this use case references the “View Attribute” use case.

Use Case Name	View Pages of a Type Tag
Actors	User
Trigger	A type tag has been clicked.
Preconditions	The type tag exists and the system is running.
Postconditions	All wiki pages, type tags, attribute definitions and attributes have not been changed.
Invariant	All wiki pages, type tags, attribute definitions and attributes are not changed.
Normal flow	1. Click on a type tag 2. Get a tabular view with all pages and the attribute values in a row and the attribute names and definitions as columns
Alternative flow	1. Search a type tag by name 2. Click on the type tag 3. Get a tabular view with all pages and the attribute values in a row and the attribute names and definitions as columns

Table 18: View Pages Of A Type Tag

Sort by Type

The rows displayed in the “View Pages Of A Type Tag” use case can be sorted by an attribute. If an attribute's value is typed, the sort algorithm has to order the items correctly, not just lexicographically.

Use Case Name	Sort by Type
Actors	User
Trigger	User clicks on a column head.
Preconditions	Attribute values are typed.
Postconditions	Rows in the Table are sorted
Invariant	Attributes are unaffected.
Normal flow	1. Click on the column head which is the criteria to sort the rows for. 2. Rows are sorted in regard to their type. If an attribute contains multiple values, the top value is taken as a reference.

Table 19: Sort By Type

System Design

A system design targeted at usability does not only focus on the use cases, it also evaluates the processes around the use cases. Therefore the following Figures show how attribute-related input data will be treated in the Hybrid Wiki system.

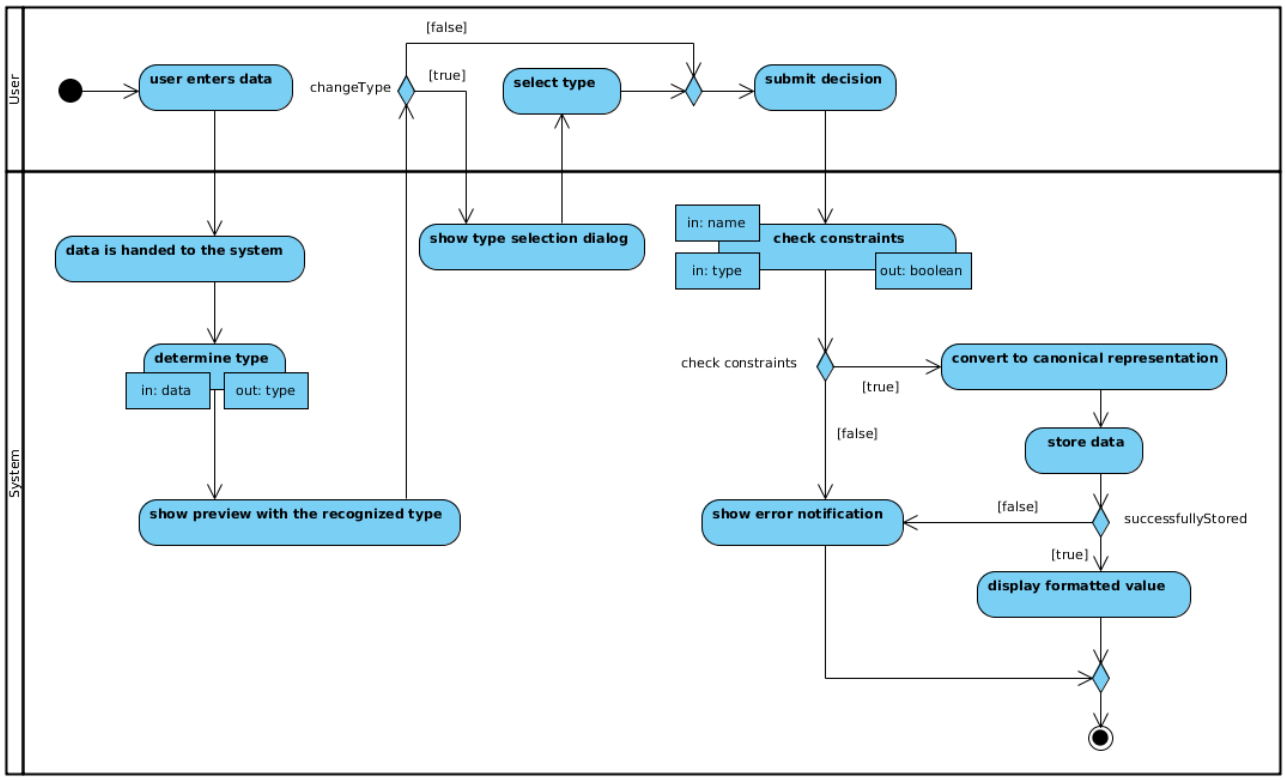


Figure 5: Activity Diagram - Attribute Value Treatment [Source: Author's design]

The scenario in Figure 5 builds on the input/edit dialog of an attribute as described in the use case "Add Attribute To Page". Once the name of the attribute has been given, the user has to supply one or more attribute values (activity: *user enters data*). The supplied data is handed to the system (activity: *data handed to the system*) after the user stops writing. Then the system evaluates which type matches the given data best (activity: *determine type*). This process is describe in detail by Figure 6. The recognized type is displayed to the user together with the value inserted (activity: *show preview with the recognized type*). A selection dialog is shown with all available types (activity: s*how type selection dialog*) the user can select from (activity: *select* type, use case: *View All Types*) and if the user decides to change the type as depicted in the use case "Specify Value's Type", s/he can select a different option from the menu. After the type has been approved or changed, the name, values and associated types of the attribute are submitted to the system by the user (activity: *submit decision*). Afterwards the system checks for constraints applicable to the given name and type (activity: *check constraints*) as described in detail by Figure 7. If no constraints are violated or set for the given name, the supplied data will be converted, otherwise an error-notification is shown (activity: *show error notification*) and the user can edit and submit the data again.

The value given by the browser to the wiki system is encoded as text and thus has to be converted into the specified type (activity: *convert to canonical representation*). This implicitly means that the value is interpreted as a number, date, text, etc. and if required, converted into an internal/canonical representation of the value. Either way, the value is stored (activity: *store data*). If the system was

not able to store the data due to the inability to convert the value to an internal representation or if the system could not write the value into the data storage, an error-notification is displayed to the user (activity: *show error notification*). In case the typed values could be saved, the formatted value is displayed (activity: *display formatted value*).

As an alternative to the mechanism described in the previous paragraph, the internal representation can be hidden from the user. In order to achieve a similar functionality, a simple flag such as “text value” can be shown as a checkbox. It indicates that the value provided by the user should be handled as text and not being interpreted as a number/date/currency value/etc. This approach would increase the usability only a little, because the user has to choose only whether to interpret the value or not instead of choosing a type. On the other hand the user cannot convert values and the user does not know which type of value s/he should enter if an attribute definition enforces a certain type on the attribute. Due to a slight increase of usability over the previously described attempt and a loss of functionality, this approach was rejected.

Determine the Type in Detail

As mentioned in the Section above, the activity *determine type* in Figure 5 is displayed in greater detail in Figure 6. The process described in this Figure begins with the retrieval of the value (activity: *retrieve value*) which is revealed by the activity *determine type* in Figure 5 as an input parameter *data*. Next the value is evaluated for a date or time format (activity: *check for date/time format*). If the given value matches the format required for a date or time, the type *date* is set as the type to be returned (activity: *set date/time as type*). After the check for date value compliance, the value is evaluated whether a format for percentage or currency is met. Assuming that the symbol for a percentage and currency value differs, the distinction is obvious. In case the value has neither been compliant with a date/time format, nor with a percentage or currency format, the value is interpreted as a number. If the interpretation succeeds the type *number* is handed as a return value to the process shown in Figure 5. In case the interpretation failed, the value is checked whether it is a key to another resource within the system. If the value is found to be a valid key, the type *reference / link* is returned. In any other case the value is regarded as a text value.

The order in which the various type formats are evaluated depends on the degree of detail a format requires from the input value. In general, a number can be interpreted as a date or time value, but if the user's locale has a different time separator symbol as the decimal and thousand separator, a number will not be mistaken as a time value. The locale also specifies three date positions: year, month and day and therefore at least two separator symbols are necessary. For this reason a number can not be taken by mistake for a date value. Still, the possibility of a number interpreted as a time value is relevant and therefore Figure 5 explains why the user is confronted with the evaluation result before saving the attribute.

During this process it is assumed the value provided is formatted as a sequence of characters as handed to the system by the browser. Currently all common browsers[18] such as the Microsoft Internet Explorer[19], Mozilla Firefox[20] and Google Chrome[21] support to submit files using an HTML-Form. For security [Münz 2006, p. 216] and type recognition reasons in this context, it is necessary to check the data handed by the browser for binary content. In the process described above, another

18 NETMARKETSHARE, Browser market share, June 2011, http://www.netmarketshare.com/browser-market-share.aspx?spider=1&qprid=0, Accessed: July 16th, 2011 3:00pm.

19 Internet Explorer – Microsoft Windows, http://windows.microsoft.com/de-DE/internet-explorer/products/ie/home, Accessed: July 16th, 2011 3:00pm.

20 Mozilla Firefox Web Browser, http://www.mozilla.com/en-US/firefox/new/, Accessed: July 16th, 2011 3:00pm

21 Google Chrome - Get a fast new browser. For PC, Mac, and Linux, http://www.google.com/chrome/, Accessed: July 16th, 2011 3:00pm.

activity has to be implemented before the *check for date/time format* activity, but this is omitted in the process shown in Figure 6, because all input fields associated with attributes in the current version of Tricia are either checkboxes or text-input fields and do not support the submission of binary data. Additionally binary content such as media content is difficult to implement and not very useful, because for functions such as structured queries and the comparison of pages which are part of the core idea of a Hybrid Wiki system, the binary type is difficult to use and therefore this thesis does not handle binary data, nor implement a binary type.

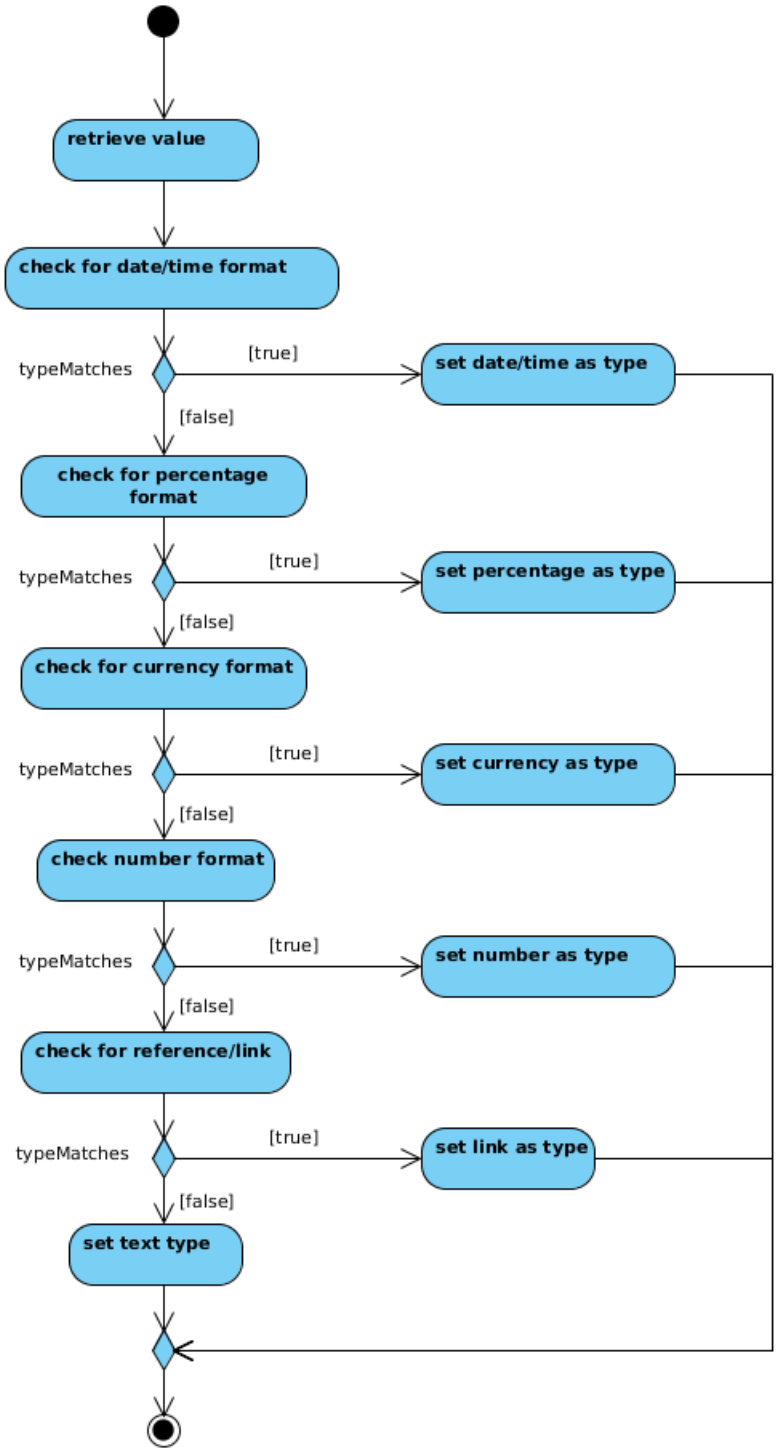


Figure 6: Determine The Type [Source: Author's design]

Check Constraints Process

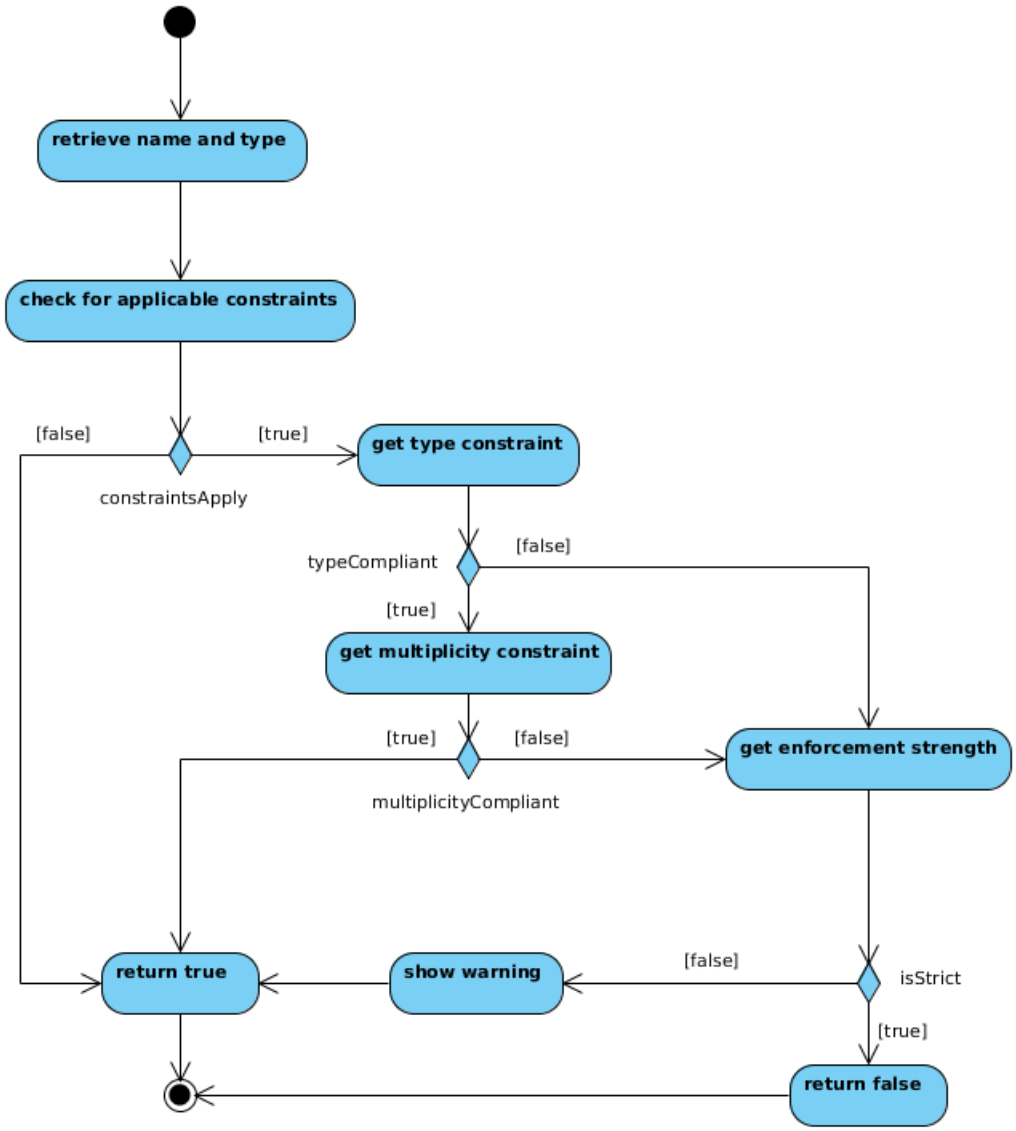


Figure 7: Check Constraints [Source: Author's design]

The activity *check constraints* in Figure 5 is described in detail by Figure 7 with its input parameters name and type as well as its boolean output. At first, the process retrieves the name and type of the attribute the user has added or edited (activity: *retrieve name and type*). Afterwards the system searches for an applicable attribute definition. If an attribute definition was found, the constraints as well as their strength of enforcement are retrieved (activity: *check for applicable constraints*). If no attribute definitions with constraints were found, the process returns true (activity: *return true*) and terminates. In case constraints apply to the attribute, the presence of a type constraint is checked (activity: *get type constraint*) and retrieved if present. If no type constraint applies or the type given by the user is compliant with the constraint, the system continues with the retrieval of applicable multiplicity constraints (activity: *get multiplicity constraint*). Analog to the *get type constraint* method and the following decision node, the *get multiplicity constraint* activity checks for the presence of a multiplicity constraint and in case one is present, it checks whether the multiplicity is compliant with the input data. If neither the type constraint, nor the multiplicity constraint matches with the given data, the enforcement strength is evaluated (activity: *get enforcement strength*). A strict enforcement of the constraints results in a

false return statement (activity: *return false*) while a loose enforcement results in a warning that the user has not supplied data compliant with the attribute definition of a certain type tag (activity: *show warning*). Afterwards or in the case all constraints are compliant with the given data, true is returned (activity: *return true*) and the process terminates.

As an alternative the activities *get type constraint* and *get multiplicity constraint* can be exchanged. The result would be the same as in the process described above, thus the order of checking the constraints applicable to the attribute is irrelevant in this case. Another variation of the process is to remove the *show warning* activity. In this case the user is notified when the attribute is displayed, however, not during the adding or editing process. In regard to the usability guidelines described in the literature research Section above, an immediate feedback is important for a usable product and therefore it was decided to display a warning message in case one or more constraints are violated, but the enforcement is not set to be strict.

Conceptual Design

From the findings gathered so far, the types, use cases and processes were clarified. The next step was to define the structure of the solution used for the implementation. Therefore the Figure above shows a conceptual unified modeling language (UML) class diagram.

From a high level perspective the structure depicted in Figure 8 on page 37 is divided into five Sections. The first Section with the classes *Attribute* and *AttributeDefinition* represent the basic entities for the Hybrid Wiki. Another Section contains the classes *Locale*, *User* and *Configuration* and is responsible for the configuration parameters and user preferences needed to format and evaluate attribute values. Besides these Sections are three Sections to fulfill the functionality to represent, format and restrict an attribute value. The Section on the left containing the classes *BooleanVT*, *NumberVT*, *UnitVT*, *PercentageVT*, *CurrencyVT*, *StringVT*, *DateVT*, *LinkVT*, *EnumerationVT* with the abbreviation VT which stands for Value Type and their superclass *ValueType* represent the types of an attribute value. All classes within this Section store an internal representation of the attribute's value and encapsulate the value by their type. To the right of the value type Section is the validator Section with its superclass *Validator*. Each class within this Section, namely the *MultiplicityValidator*, *BooleanValidator*, *NumberValidator*, *PercentageValidator*, *CurrencyValidator*, *StringValidator*, *DateValidator*, *LinkValidator* and the *EnumerationValidator*, represent a constraint applicable to an attribute value. As shown in Figure 7, the process returns a warning message or the complete validation process returns an error notification as shown in Figure 5. Each validator contains a message which can be displayed to the user in case the constraint is violated. In between the validator Section and the configuration Section is the formatter Section with its *Formatter* superclass and the *NumberFormatter*, *UnitFormatter*, *DateFormatter*, *LinkFormatter* and *EnumerationFormatter* classes derived from it. The formatters main purpose is to format values according to the user's locale as well as evaluate the user input for a certain type.

The relation between an attribute and its definition is described in Figure 1. The name of an attribute and the name of the attribute definition must be equal to associate them with each other. An attribute can have multiple attribute definitions from different type tags and an attribute definition refers to multiple attributes from different pages. Wiki pages and type tags are not shown in Figure 8, because their relations to the attribute and attribute definition is already described in Figure 1. Every attribute references an ordered list of typed values. The superclass for all types is *ValueType*. Derived from it are specific types which contain the actual value in an internal representation. The class *BooleanVT* stores its value as a primitively typed boolean value. Boolean user inputs are

validated by the class *BooleanValidator*, but a formatter class is not needed in this context, because the screen representation of the value only requires a primitively typed boolean value. However, *NumberVT* requires a formatter class called *NumberFormatter* in order to format the internally stored primitive floating-point number value as a *String* regarding the user's preferences and the precision which describes the amount of decimal places. In addition, *NumberVT* is referenced by a *UnitVT*. This type is the abstraction of all specific unit values such as percentage, weight or even currency values. Unit values consist of a numeric part represented by the *NumberVT* reference and a symbol. For example, “12 %” is separated into the numeric part “12” and the symbolic part “%” which can be interpreted separately. Nonetheless every subclass of *UnitVT* needs its own validator to check for the correct format including the symbol. Only the *UnitFormatter* class can be generalized by parameterizing its methods to use a specific set of symbols. *PercentageVT* and *CurrencyVT* are depicted in Figure 8 as subclasses of *UnitVT*. Their validators *PercentageValidator* and *CurrencyValidator* can be used by many *PercentageVT* and *CurrencyVT* objects while the validator objects act as the constraints applicable to the composite values. Similar to the *CurrencyVT* and *PercentageVT* classes is the *DateVT* class with a value consistent of an internal date presentation and a precision. The precision is enumerated and describes how detailed the date and time values have to be displayed. The enumeration contains the components of a date such as year, month, day, hour, minute, second, fraction of a second as well as the combination of all components which is called “complete”. Optionally other components like the day of the week or the timezone can be added to the enumeration. For example, if the data given by the user in plain text format reads “2010-12-24”, the precision “month” is set and it is assumed the order of the date is year-month-day, only “2010-12” is the value in return. In case the precision is set to “second”, the result is “00:00:00” given the time separator is a colon. If the precision is set to “complete” a full date time value like “2010-12-24 00:00:00.0” is returned. Upon setting a new value for the *DateVT* object, the precision is determined and saved with the internal date representation. The *DateValidator* fulfills a similar functionality as the *PercentageValidator* / *CurrencyValidator* by simply acting as a constraint. In contrast to the *NumberFormatter* which is also used by the *UnitFormatter* to format the numeric part of a unit, the *DateFormatter* formats all composite values according to the user's preferences, but without utilizing any other *Formatter* derivatives. Also the *LinkFormatter* and *EnumerationFormatter* do not use other *Formatter* subclasses to format the values handed to them by their value type classes *LinkVT* and *EnumerationVT*. Alike all other classes *LinkVT* and *EnumerationVT* require to have their own constraint implementation represented by the *LinkValidator* and *EnumerationValidator* class. Every validator class depicted in the Figure constraints none, one or many value type objects while each of the value type objects is only constraint by one validator. Still value type objects can be validated by more than one validator, but not in the scope of one attribute definition. Like a validator is restricted to an attribute definition, the formatter classes are restricted to look in the user preferences for the correct format settings. Therefore the class *Locale* holds all required settings and its objects can be referenced by an user or the system configuration. The system configuration is embodied by the class *Configuration* and has to reference exactly one *Locale* object as the fall-back locale for all users who have not set their locale individually. Users are modeled as objects of the *User* class and reference none or one *Locale* object. The *Locale* class itself contains a character to separate decimal places (*decimalSeparator*), another character to group thousands within a number (*thousandSeparator*), a character to separate the single values of a date (*dateSeparator*) and analog a character for time separation (*timeSeparator*). The order of a date as described above is stored in an array with the length of three (*dateOrder*): one place for the position of the year, one for the position of the month and one for the position of the day. Furthermore the *Locale* class comprises two boolean values whether the day and month will be displayed with two digits (*twoDigitDayMonth*) and analog whether the year will be displayed with four digits (*fourDigitYear*). Additionally the *Locale* object determines whether a unit is placed in front of a value (true) or

behind it (false), for example, if the value of *unitBefore* is true, a currency value is displayed as “EUR 123.00”, otherwise the value is displayed as “123.00 EUR”.
Lastly, the *StringVT* has no Formatter class and therefore does not use *Locale* objects. Even a validator is only needed to define attribute definitions to explicitly restrict attributes to contain values typed as *String*. Any unrecognized user input value is handled by the system as a *String* type and therefore no restrictions apply.

Alternatively to the system design depicted in the previous paragraphs, all formatter classes can be implemented in the *ValueType* subclasses. The *ValueType* objects could directly access the *Locale* objects and less classes would be needed to fulfill the same purpose. However, this results in bigger classes and less maintainability, because the separation between the encapsulation of the value itself and the formatting is lost. If the formatting algorithms have to be exchanged, the value classes themselves will have to be changed whereas if the formatter classes are separated from the value classes, the new algorithm can be a subclass of the previous formatter class and only very little code will have to be altered in the *ValueType* subclass. Another aspect is adding more configuration parameters to the *Locale* class. In case the formatter is included in the *ValueType* subclass every change of the Locale class results in a change of the value type while in the separated approach new configuration parameters can be added to the locale and implemented in the formatter classes or even generalized for all formatter classes.

In detail, the representation of numbers can be divided into a representation for integer numbers and one for floating-point numbers. Since the formatting process is similar for both types and the usability is worse due to the fact that the user has to decide whether to use an integer or floating-point number, the approach shown in Figure 8 was chosen. Like the *NumberVT* can be split, it would also be possible to split the *DateVT* class into one representing date values, one for time values and other classes for date-time combinations. In general all these classes would implement another dimension of the date value and therefore the approach was chosen to combine them in one class and add a precision dimension. The definite implementation of this dimension can be replaced by a representation for date, time and the date-time combinations, but the enumeration approach had been chosen due to its variability. As this implementation can easily be misunderstood by programmers unfamiliar with the matter, an accurate documentation of the code is necessary.

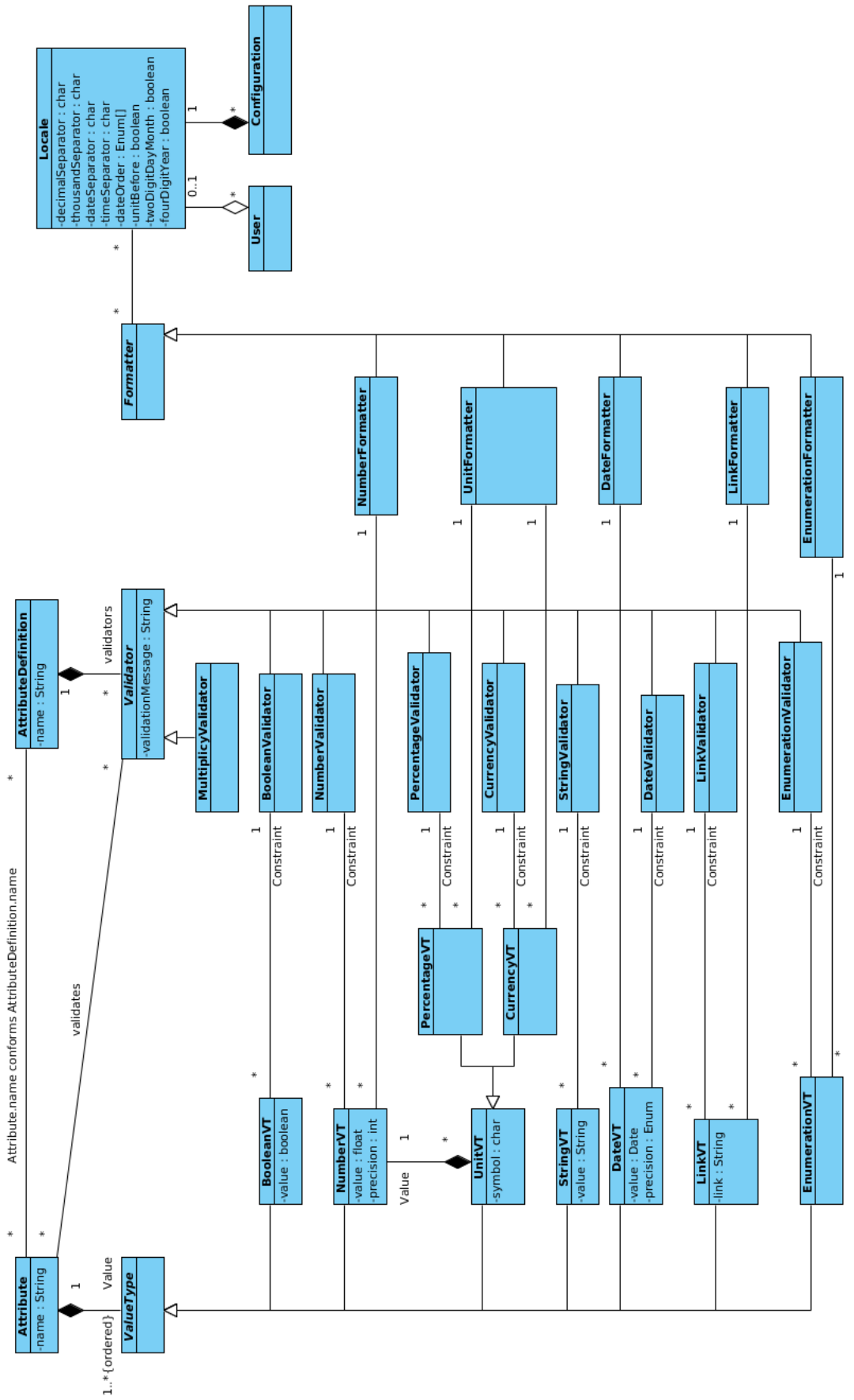


Figure 8: Conceptual System Design [Source: Author's design]

Tricia Specific Design

Since the conceptual design was incomplete to use for an implementation, a mapping of the conceptual design to Tricia was needed. The Figure below depicts the design details for the implementation process. Only a subset of the classes in Tricia are displayed in the Figure in order to focus on the most important parts of the design.

Classes colored in gray were already present in the system, classes in blue were added to the system within the scope of this thesis as well as the enumeration marked in green. The yellow packages divide the design into five distinct partitions. On the left is the package *de.infoasset.code.assets.hybrid* which mainly contains the *ValueType* hierarchy of Figure 8. To the right of this package is an extension called *de.infoasset.core.assets.hybrid.typeConstraints* which was, in contrast to the previously mentioned package, not present in the system. Also the package *de.infoasset.core.hybrid.typeFormatters* was not part of the system, but it was planned to be added. The internationalization package *de.infoasset.platform.services.internationalization* and the package *de.infoasset.core.assets.group* containing the user, group and system group entities were already present, but not yet associated with each other. Another major difference is the names in both Figures. The *HybridPropertyDefinition* class corresponds to the *AttributeDefinition* class in the conceptual design. The class *HybridProperty* represents the attribute and the *ValueType* class was mapped to the abstract *HybridPropertyValue* class. All “VT” abbreviations were exchanged for “Value” and the date-precision enumeration was named *DateTimePrecision* and added to the design. Additionally also the *RichStringValue* class was added for a complete overview of the types present in Tricia. In the *typeConstraints* package, all former “Validator” terms were exchanged for “Constraint” in order to stay consistent with the classes already present in the system. In addition, the superclass was renamed to *AbstractTypeConstraint* instead of *Validator* and the *StringValidator* became the *TextConstraint*, because the characters in front of “Constraint” are used as the types' names which are presented to the user. Within the *typeFormatters* package, the class *Formatter* was renamed to *AbstractFormatter* for consistency reasons. In Figure 9 on page 40 the two packages on the right were adapted from the current Tricia design except for the associations between the *Principal* derivatives and the *UserLanguage*. In comparison with the conceptual design the *Configuration* class is missing. In Tricia the system configuration is stored in XML files which are generated from annotations in the code. Thus the configuration can be directly applied to the *UserLanguage* class. Another noticeable difference in the internationalization package is the distinction between an user (*Person*), a group of users (*Group*) and a system group (*SystemGroup*) such as administrators or unknown users. Since their superclass *Principal* is also used without the locale context, the locale field is not pulled up into the *Principal* superclass. Furthermore this package contains a class called *UserLanguages* which manages the *UserLanguage* objects.

The *UserLanguage* class roughly contains the same fields as the *Locale* class in the conceptual design. However, due to the configuration process, the data types were slightly altered. Methods such as *getDefaultDateTimeOrder()* which returns the system's default date order and *parseDatePosition(pos : String)* which is needed for the configuration process, were added. All getters and setters are not displayed in this model for readability reasons.

In the *typeFormatters* package, the superclass for all other classes contained in this package is *AbstractFormatter*. The main purpose of the formatters is to receive a value as a parameter, to format the value according to the *UserLanguage*, and to return the formatted value as a *String* object. These formatting methods are static, because it is not required for them to keep any state after the execution of the method has terminated. Therefore a class like the *UnitFormatter* is rather a method collection, than an object scheme. Unlike the other formatter classes within this package, the *UnitFormatter* defines two unusual methods. One of them is *splitParts(v : String, symbols :*

String[]) : String[] which separates the given value named *v*, for example the user input value, into a numeric part and a symbolic part. A set of valid symbols (parameter: *symbols*) helps the method to determine where to split and whether *v* is dividable. The other exceptional method is called *isSymbolValid(sym : String, symbols : String[]) : boolean*. It takes a *String* object and an array of valid symbols and checks whether the symbol is contained in the array. If it is contained in the array, true is returned, otherwise the result is false.
In contrast to this exceptional formatter, the class *DateFormatter* offers methods to format a date, time and date-time values as well as it parses a *String* object for date, time and date-time values. The method *formatDate(...)* and *formatTime(...)* are convenience methods for the *formatPrecision(...)* method. Analog is the *parseAnyDateTime(...)* also a convenience method which calls the other parse-methods in order to determine the *Date* result. In addition to the depicted classes were the *LinkFormatter* and *EnumerationFormatter* classes shown in the conceptual design, but removed from the implementation model, because the formatting of these values is handled in another class within the hybrid module of Tricia.

Merely changed from the conceptual design was the *typeConstraints* package. Similar to the *typeFormatter* package, one class is the superclass for all other members. In this package, the class is called *AbstractTypeConstraint* and coincidently, it is the only one modified. Instead of having the validation message as a field as described in the conceptual design, the message is generated in a method called *validationMessage(o : Object) : Message* which takes the attribute values as an input and returns a message, if the validation process as described in Figure 5 and 7 requires to show one, otherwise *null* is returned. Additionally four more methods were added to the class *AbstractTypeConstraint* which are already implemented in Tricia, but helpful to understand for further explanations.

- *getRepresentation(clazz : class) : String* cuts the “Constraint” trail from the class name and returns only the part without the “Constraint” term. The method is called on creating the types list for the user interface.
- *GetTypeProperty() : TypeProperty* is a convenience method to return the type set for the *HybridPropertyDefinition* object internally referenced and also returned by the method *getHybridPropertyDefinition() : HybridPropertyDefinition*.
- *GetAllTypeConstaintClasses() : List<Class>* returns a list of classes defined to be regarded as types available to the user.

Primarily however, the *hybrid* package is the core of the design. For readability reasons, the relevant subset of the implemented methods and fields is shown. As depicted above, the class *HybridPropertyDefinition* corresponds to the *AttributeDefinition* class in the conceptual design, but was enriched with the crucial fields such as *isStrict* and *multiplicity*. Although the methods *getTypeConstraint()* and *getFailingValidators(...)* had already been implemented, they are shown in the model, because of their reference to the term “validator” and the *AbstractTypeConstraint*. As the annotation of the relation between the *HybridPropertyDefinition* and the *HybridProperty* suggests do the *HybridPropertyDefinition*'s name attribute and *HybridProperty.getSimpleName()* method correspond in case an attribute definition has been created for the attribute. Since every attribute has multiple ordered values, the *HybridProperty* class includes methods to get and set the values which are represented by the *HybridPropertyValue* subclasses. Except for the *EnumerationConstraint* and the *BooleanConstraint*, all other constraints have corresponding value types. However, in both cases the values are represented by the *StringValue* and already functioning. Analog, the *RichStringValue* does not have a type constraint and is not regarded any further in this thesis due to the fact that rich string values can be represented utilizing StringValue at the moment. Also the *LinkValue* is not discussed in detail, because of its substantiated implementation.

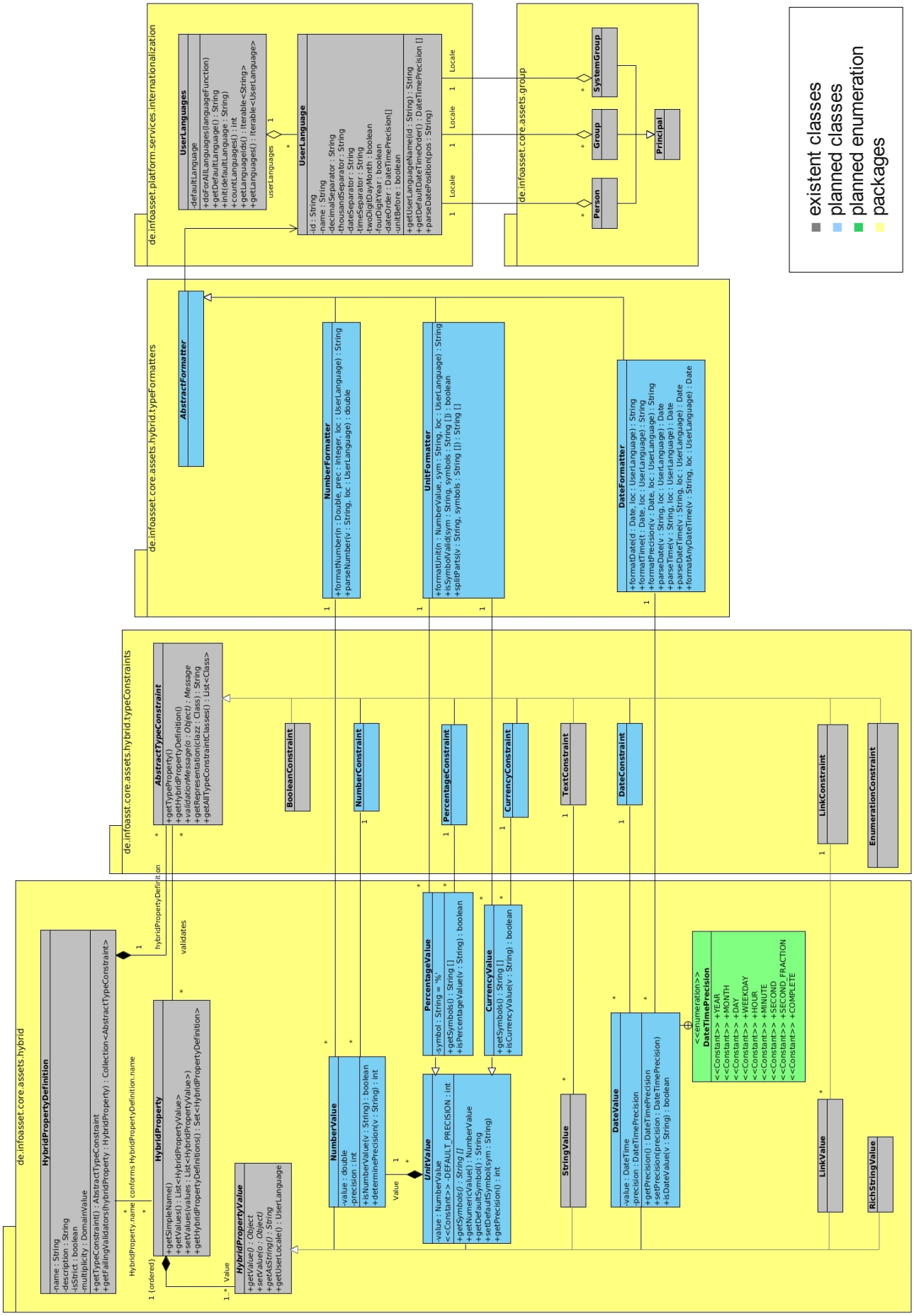


Figure 9: System Design [Source: Author's design]

As in the conceptual model, the *DateValue* class includes two fields: a value of the type *DateTime* and the precision as a distinct value of the *DateTimePrecision* enumeration. The manifestations of the latter are shown as constants in the enumeration colored in green. As all other subclasses of *HybridPropertyValue*, the class implements the method *getValue() : Object* which is meant to return the stored value, *setValue(o : Object)* which sets the given object as the stored value and *getAsString() : String* which returns the formatted stored value. Additionally, *DateValue* calls the *HybridPropertyValue.getUserLocale()* method to retrieve the current user's locale settings. Similar to the *NumberValue*, the *PercentageValue* and the *CurrencyValue* class, a static method is implemented which determines whether a given string can be set as a valid value for the called class. In case of *DateValue*, the method is called *isDateValue(v : String) : boolean*. The *is...Value()* methods were not generalized and moved to *HybridPropertyValue*, because only half of the subclasses offer the method and thus it would be misleading if they were provided in the superclass.

A detailed look at the *NumberValue* class shows the internal value representation of a numeric value as a double value instead of a floating-point value as in the conceptual design. The precision field was adopted from the conceptual design and the static *isNumberValue(v : String) : boolean* method inspects the input for its compliance with the number type. Similar to the *DateValue* class, the *NumberValue* class includes a method to determine the precision of the input data. The calculated amount of decimal places is later required to format the numeric value correctly.
References to the *NumberValue* class are used by *UnitValue* to store and format its numeric part. In contrast to the *NumberValue*, however, the *UnitValue* has the ability to define the precision. In case the precision is less than the given decimal places, they are rounded to the closest defined decimal place. By default, the number of decimal places is set as the *DEFAULT_PRECISION* constant which is of type *int* (integer). The method *getPrecision() : int* returns the constant by default, but it can be overridden by the subclasses. For example, the currency class will most likely need two decimal digits for its values and therefore the *getPrecision()* method will need to return the integer value two. While the *CurrencyValue* class has the valid symbols for a currency like “EUR” or “€”, the *PercentageValue* class has a fixed set of symbols such as “%” or “percent”. In both cases the *getSymbols() : String []* method has to be overridden in order to return the valid symbols for this type. Within the *UnitValue* superclass *setDefaultSymbol(sym : String)* sets the symbol used to format the value and *getDefaultSymbol() : String* retrieves the default symbol.

The approach to generalize the behavior of unit values was chosen due to its extensibility. All unit values have a numeric part and a symbolic part, but just the latter is unique for each unit value type. In addition, the numeric part is always a number and therefore the *NumberValue* class was chosen to represent these values and take care of the numeric formatting. During the design development the precision issue was discussed in detail. *Double* values are not as precise as they seem and therefore an internal representation of a numeric value as a *java.math.BigDecimal* object was discussed. The reason why this idea was rejected, was because of its worse performance and the missing use case. For scientific calculations the system is inadequate, but since this is not the perspective of Tricia, it was chosen to stick to the *double* value representation. Another topic discussed was to merge the *typeConstraints* and *typeFormatters* packages into the *hybrid* package, because of the dependencies and close ties in between the packages. A loss of logical separation and one big package would be the result and because the effects of separation are not noticeable for the user, it was decided to keep the separation for better maintenance. Yet another alternative was in discussion for the *internationalization* package. Since the *UserLanguage* class had already been implemented, a class called *Locale* can be derived from it and used by the derivatives of *Principal*. This solution was rejected, because it creates double as many objects for the same purpose and an object management either within the *UserLanguage* or the *UserLanguages* class would have been required.

Graphical User Interface

Now that the system design is mapped to Tricia and an internal representation is found as well as the layout for formatting and constraints is set, the graphical user interface has to be designed. In order to create an interface with high usability, the criteria for good graphical user interface design, as described in the literature research Section, have to be met. At first a short analysis of the current interface will be depicted, then the planned changes will be discussed.

Create a Wiki Page with Attributes

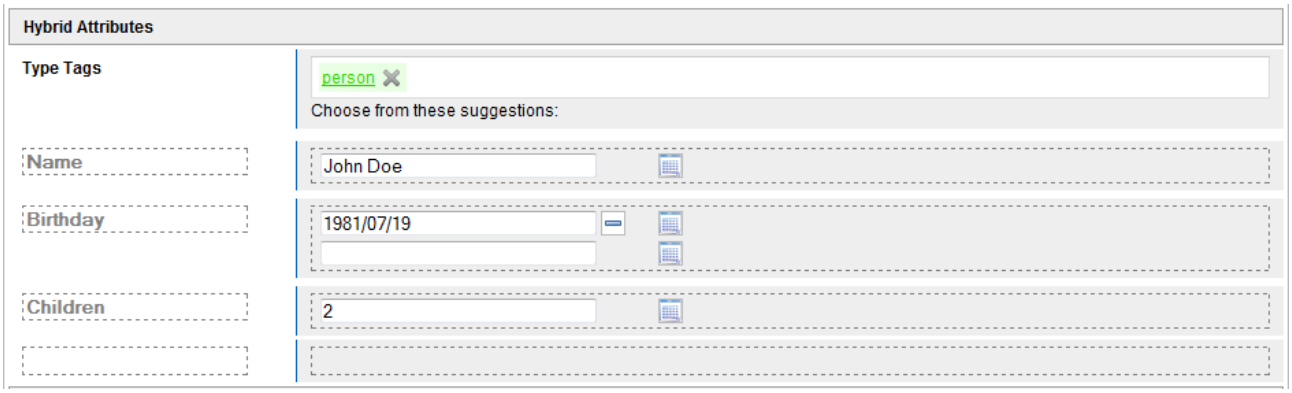


Figure 10: Create Wiki Page With Attributes
[Source: Exemplary screenshot of Tricia taken by the author]

Figure 10 shows the “Hybrid Attributes” Section of the screen to create a new wiki page. On top the user can optionally enter a new type tag or choose one or more from the existent type tags. If a type tag was selected, attribute suggestions are made. If no type tag was chosen, only one row like the one at the bottom of the Figure, will be shown. In the left area of the screen the user enters the name of the attribute and in the right dashed area an input field comes up by clicking on the area. Next to the input field a calendar symbol is shown which opens a calendar if the user clicks on it. In between the calendar symbol and the input field, a minus-symbol is displayed. A click on it removes the value from the list of values of the attribute. With the current interface, the user cannot change the type of the input, because all values are either a text, link or boolean value and all are stored as text. The values are interpreted on creating the output and thus the typing cannot be influenced by the user. As a solution to this problem, a selection of available types will be shown on the right of the calendar icon. By default the text type is shown, but the user input is interpreted after s/he stops typing and if the user input was interpreTable, as for example a date or number, the selection menu changes to show “Date” or “Number” as a type. If this type is not in the user's intention, s/he can select another type in the selection menu. If the type selected is not “Date”, the calendar icon is removed from the panel. The interpretation procedure is described in detail in Figure 6 and gives the user an immediate feedback as postulated in the graphical user interface guidelines. It also suggests to the user, s/he is in control of the input interpretation.

Display Attributes on Wiki Page

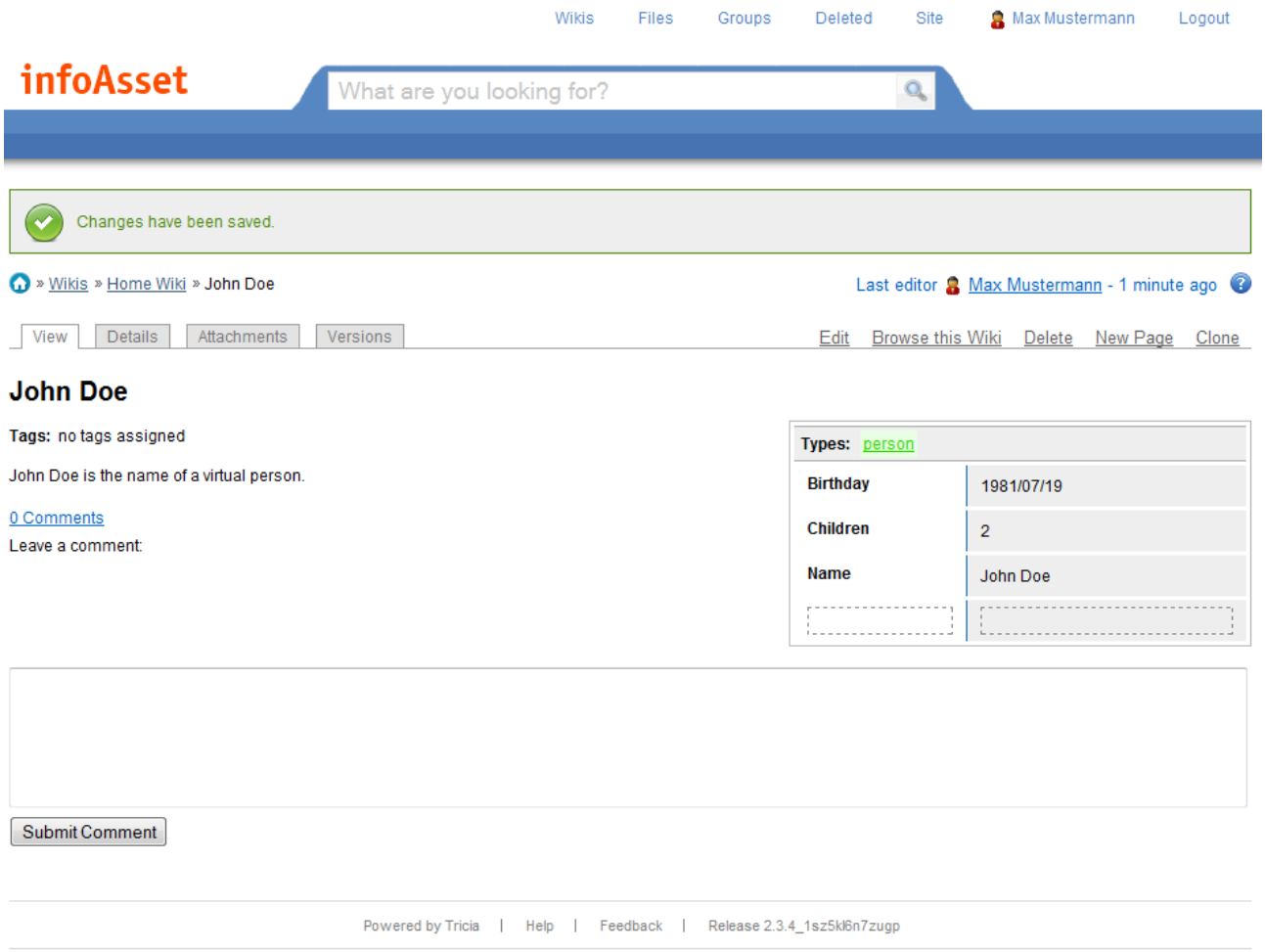


Figure 11: Attributes On Wiki Page
[Source: Exemplary screenshot of Tricia taken by the author]

The attributes added to the wiki page in Figure 10 are displayed on the page as shown in Figure 11. On top of the area reserved for attributes are the type tags. A click on a type tag shows a tabular overview as depicted in Figure 16. Below, on the left side is the attribute name displayed and on the right side the attribute values. Since the values are presented as simple text values, the format depends on the user input. An English user, for example, inputs the dates in a different format than a German user. Misunderstandings and a loss of productivity is the result as depicted in the introduction. In order to reduce these drawbacks, the thesis strives at a unified view of attributes. The user's locale determines the settings used to format every value according to its type. Thus a similarity in appearance is generated as claimed in the literature.

On the example page shown in Figure 11 the user can access his settings by clicking on his name, shown on the top of the page, left to the “Logout” link. Also in this line are links to access other wikis, files, deleted pages and sites. The “Groups” link allows to show the available groups and if privileges are available, these can be edited. In the center below is a search field. Searches apply to various entities such as wiki pages, tags, type tags, attributes and so forth. Below, a message is shown “Changes have been saved.” In case the page has not just been added or edited by the person viewing it, this message will not appear. Below the message Section on the left is the path to the current wiki page. On the right information about the last wiki page editor is shown. In the row below the editor on the left are tabs for the wiki page content (“View”), details about the page

("Details"), file attachments ("Attachments") and the change history ("Versions"). On the right links to edit the wiki page are shown as well as a link to browse pages of the current wiki, a link to remove the current page, one to create a new page and one to clone the page. In the lower part of the wiki page a text area is shown where comments can be added to the page. The footer on the bottom of the page contains additional information about the software and support.

Edit an Attribute on a Wiki Page

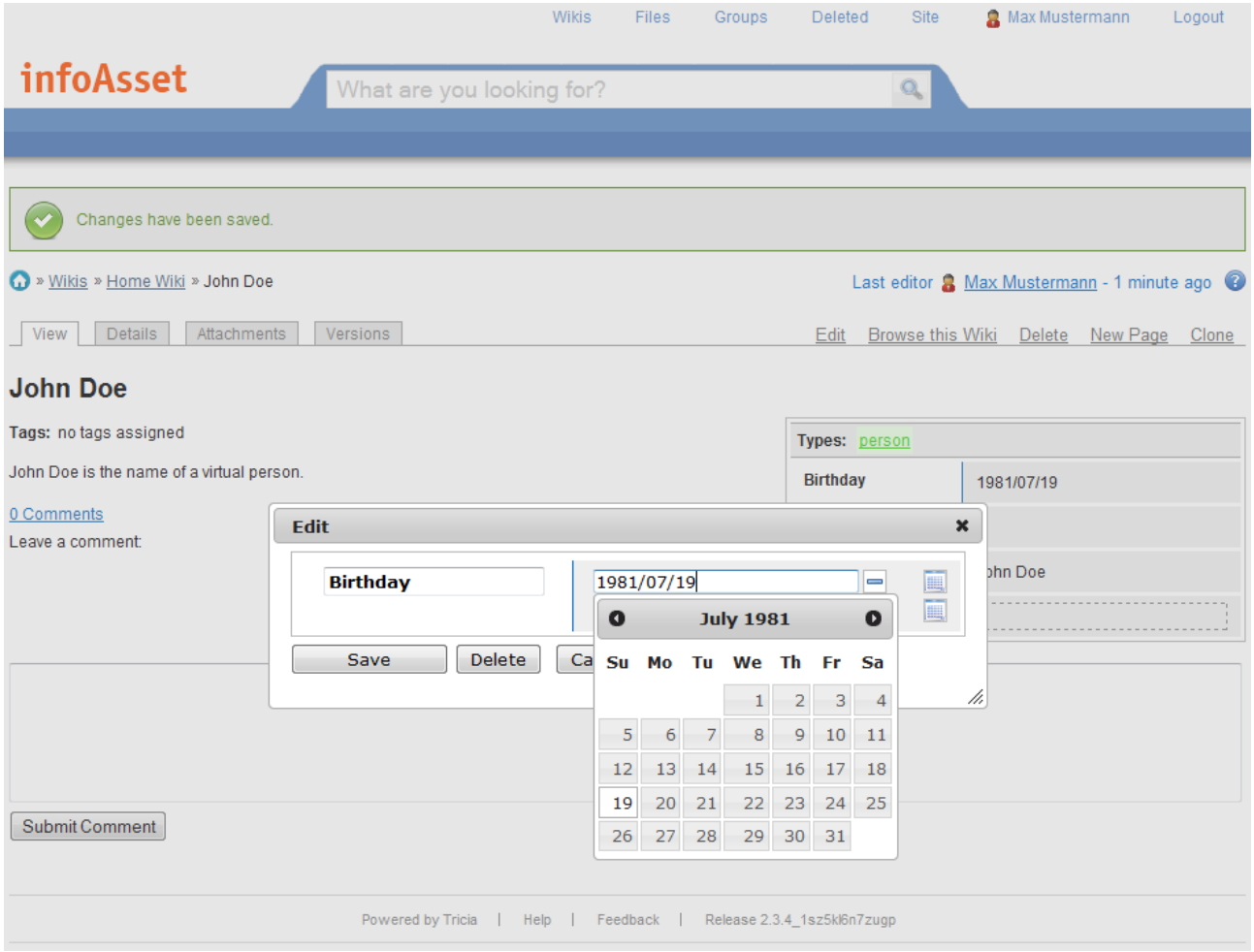


Figure 12: Edit Attribute On Page
[Source: Exemplary screenshot of Tricia taken by the author]

To edit an attribute on the page one hovers over the attribute's name or value and clicks on the pen icon displayed next to the field. An overlay as shown in Figure 12 is displayed showing the attribute name on the left and the attribute values on the right. As described above, a click on the calendar icon opens a calendar and then the user can choose a required date. The minus-symbol on the right of the input field removes the value from the list of the attribute's values. As depicted before, the user has no control over the value interpretation and type. Therefore Figure 13 shows a possible solution, similar to the one presented when a wiki page is added or edited.

Since the draft shown in Figure 13 is only for that draft's presentation, the name of an attribute as well as additional values are left out. The user types a value in the input field, the value is evaluated afterwards and the recognized type is set to be visible as the selected item in the selection menu on the right. In case the type is "Date", the calendar icon in between the selection menu and the input field is shown, otherwise it is removed. If the type recognized by the system is not in the user's

intent, s/he is able to change the type by clicking on the selection menu where all types are shown as a list. A click on the “Submit” button results in an attempt to save the input value with the given type, but if the type does not correspond to the input value, the value is saved as a text and a warning is issued. As mentioned before, this mechanism is applied to all cases of adding and editing an attribute in order to create the functional consistency needed for a good user interface.

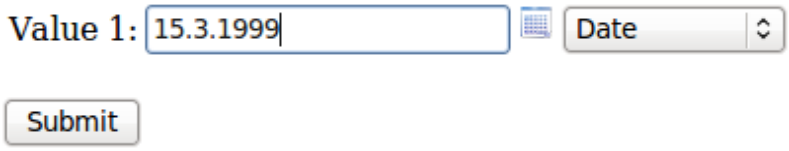


Figure 13: Attribute Value Draft [Source: Author's design]

During the discussion other solutions were evaluated, too. The approach described above does not contain a preview of the formatted value. Therefore another field has to be added either in front of the selection menu or behind it. This adds more functionality, but overloads the GUI. Thus it was decided not to add the preview-field in the scope of this work. Instead of the selection menu, other representations were discussed. Figure 14 shows two other possibilities:

- The attribute value of “goal reached” is recognized as a date value and therefore the writing of “(Date)” is shown. This approach saves space and also gives an instant feedback, but in return, long input field values reach underneath the writing which confuses unexperienced users. A click on the writing opens up a list of all available types the user can choose from. In general, the approach is not as user friendly and intuitive as the one described above, although some space is saved which makes it an alternative for small screens.

- Instead of a text representing the type to the user, a symbol is shown. The attributes “main goal” and “start date” demonstrate this approach. The input “success” was interpreted as text and therefore the date symbol changes to show a symbol for text. In case of the “start date” attribute, a number icon is shown. A list of all types is displayed, if clicked on the icon where the type can be changed. In case a date is chosen or recognized, a calendar is opened if the user focuses on the input field. The approach also saves space, has the same functionality and can increase the appeal, but users already familiar with the system, expect to see a calendar when they click on the calendar, not a list of types. The behavior of the system would be changed and the usability is exchanged for appeal and thus this approach was rejected.

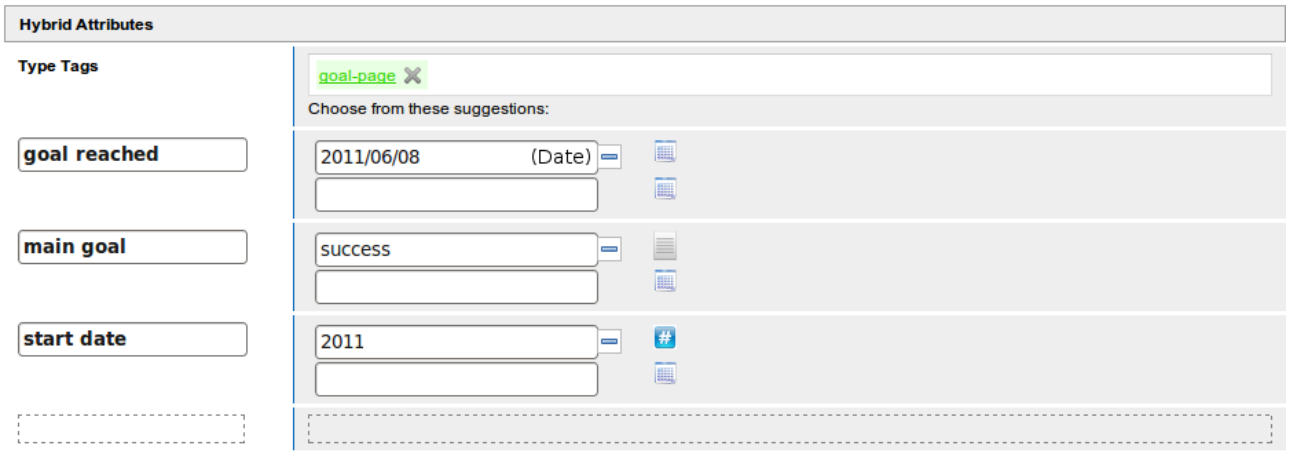


Figure 14: Attribute Value Alternatives
[Source: Exemplary screenshot of Tricia taken by the author]

Add or Edit Attribute Definition

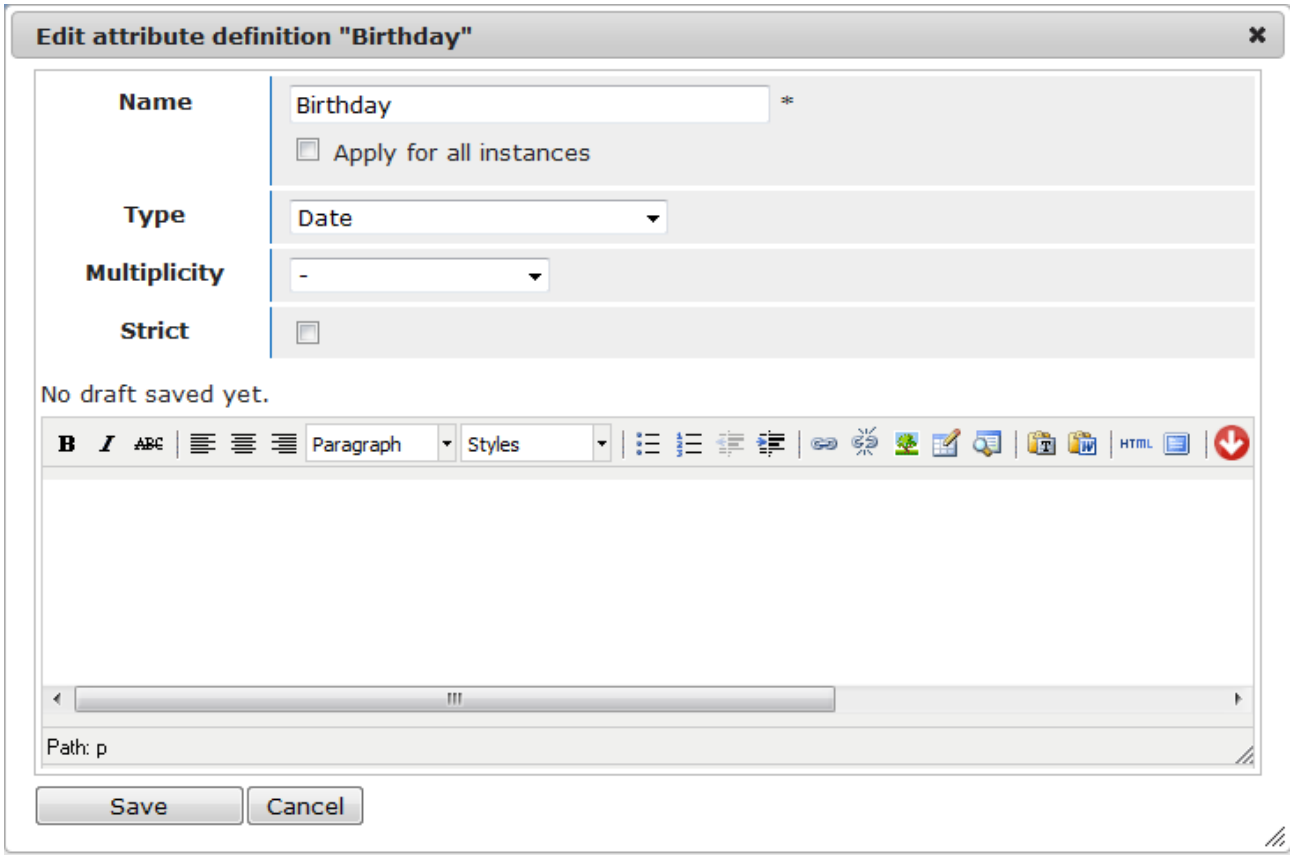


Figure 15: Create New Attribute Definition
[Source: Exemplary screenshot of Tricia taken by the author]

The dialog to create or edit an attribute definition is shown in Figure 15. Below the title of the dialog, the name of the attribute definition can be chosen and it can be set whether to apply this name for all instances. Below the type, the multiplicity and enforcement strength is set and finally a note can be added. Except for the new types such as “Number”, “Percentage” and “Currency” added to the “Type” selection menu, the dialog is not changed. Other types such as “Date”, “Boolean”, “Text”, “Link” and “Enumeration” are already present in the list. The selection menu is the appropriate control, because the types are fixed. A text input field requires the user to remember the available types and is therefore inappropriate.

Tabular View of Type Tag Associated Wiki Pages

Figure 16: Tabular View Of Type Tag Associated Pages
[Source: Exemplary screenshot of Tricia taken by the author]

An overview of all wiki pages associated with a type tag can be displayed by clicking on a type tag. This view is depicted in Figure 16 with the wiki page shown in Figure 11 as the content. Since only one row is in the Table, the navigation of pages, as shown in the upper right corner of the Table, is merely populated. Next to the navigation is a search field to filter the contents. Below is the Table header showing all attributes as columns. Exceptional is the “i” symbol in the “Birthday” column which implicitly states that an attribute definition is attached to this column. The values shown in the Table are unformatted. To create a similarity in appearance, especially in regard to the view of the attributes on a page as depicted above, the values will be formatted according to the type and the user's locale. A click on an attribute's edit icon results in the same edit dialog as a click on the edit icon of an attribute on a wiki page.

User Locale

Figure 17: User Profile [Source: Exemplary screenshot of Tricia taken by the author]

One form of interface customization which is demanded by the literature, is to apply a locale. In order to do so, the locale of each user has to be determined and the user has to be empowered to change his or her locale. The current user settings are shown in Figure 17. On top the real name of the user is shown, below his or her e-mail-address and the tags associated with the user are shown. In the Table below the tags, are the group memberships displayed together with their state and a comment. The checkbox below the Table sets the profile's visibility for the public as described in the text below. The last login is on the bottom of the settings screen.
A new parameter called “Locale” is added to the settings. Its value is a selection menu with the currently set value selected. In case the user wants to change his or her locale, s/he simply selects another locale from the menu and confirms the selection by saving his or her profile changes.

Configurator

Similar to the locale applied to an user, one default locale has to be applied to the whole system. A graphical configuration tool is already built into Tricia. The tool reads in data models and generates the configuration tree as shown in Figure 18. The language settings can be found navigating to “Main” → “services” → “userLanguages”. By selecting “defaultLanguage” in the tree, the information associated with the item is shown on the right of the tree is displayed. In the input field the default language can be changed by entering a different language identifier and selecting “Set” afterwards. The identifier has to match one of the “UserLanguage” *id* values shown within the “userLanguages” branch of the tree. In addition to the configurable parameters show in Figure 18, the parameters *decimalSeparator, thousandSeparator, dateSeparator, timeSeparator, dateOrder, fourDigitYear, twoDigitDayMonth* and *unitBefore* will be shown as described in the *Locale* class of the conceptual design. To create a new locale, the “userLanguages” tree item with the folder icon has to be selected. A list of available locale templates will be shown and “New” can be clicked to create another entry within the tree structure. An *id* (identifier) and a *name* is mandatory, all the locale settings are preset with default values. The locales within this structure are the ones, an user can choose from.

All Figures above show the most important graphical interface changes, still some other minor changes are necessary to be made. Such changes include to add a locale parameter to the group and system group settings with a similar GUI as shown for the user settings and to adjust the position of the minus-symbol in the edit attribute dialog in between the input field and selection menu if the calendar icon is removed due to a type change.

Figure 18: System Configurator
[Source: Exemplary screenshot of Tricia taken by the author]

Implementation

Having completed the planning process, the implementation of the solution as described in the previous Chapter was next. At first, the implementation methodology is described briefly together with the development phases. Then each development phase is depicted in detail and finally the outcome is presented.

Implementation Methodology

So far, the development process followed a linear model, similar to the waterfall model. [Hansen/Neumann 2005, p. 267] At first an introduction was given and the problem was stated. A description of the use cases and requirements was created in order to derive a conceptual system design from these documents. The system design was specified for the Tricia platform and the graphical user interface implementation lined out. Within the implementation process however, the linear model is enriched with methodology from agile development called Test-Driven Development (TDD). In contrast to traditional programming where a test is usually written after the completion of the application code, tests in TDD are written before the application code is created. The idea is to write a test which verifies the new functionality that has to be developed. Then it is assured the tests fail before the application code is implemented. Afterwards just as much code is implemented to make the test pass and more tests are added to the code so that the code improves just enough to fulfill the requirements. In the end, every function in the software has a test. Once all code is tested and in case all tests pass and correspond to the requirements, then the software will be of high quality. [Astels 2003, p. 6-8]

The development associated with this thesis was divided in three phases. In phase I the configuration of Tricia was adjusted, in phase II the structure of the *hybrid*, *typeConstraints* and *typeFormatters* packages was set up and tests were written for the following implementation while in phase III the graphical user interface was adjusted. Especially in phase II, test-driven development was deployed, because it is the most critical to cover the functionality of the additions. In phase I and III, mostly adaptations had to be implemented and therefore a systematic testing approach after the development was chosen.

Phase I – Configuration

In Tricia, the system configuration is saved in XML files which are generated by a tool called "Configurator". The parameters to configure are annotated in the Java classes using the "@Property" annotation [Büchner/Matthes 2006, p. 7]. A plug-in for Eclipse[22] examines the annotated fields and compiles a XML document named *repository.xml* which serves as the input for the configurator. When the configurator is started, the *repository.xml* is read and displayed in a tree structure as shown in the Section "Configurator" above. Once the administrator has completed the configuration, s/he can save it by clicking on "Save this Configuration". An output XML called *configuration.xml* is created and stored.

The task was to add more parameters for each *UserLanguage* in order to configure them with the configurator and to associate the *Principal* derivatives with an *UserLanguage* object. At first the *UserLanguage* class got a *decimalSeparator*, *thousandSeparator*, *dateSeparator*, *timeSeparator*, *fourDigitYear*, *twoDigitDayMonth* and *unitBefore* field. Unluckily the configurator did not support to display a configurable fixed-size array for the *dateOrder* field. Thus the *dateOrder* field was split into three parameters *dateOrder_position1*, *dateOrder_position2* and *dateOrder_position3*, each

22 Eclipse - The Eclipse Foundation open source community website. http://www.eclipse.org/, Accessed: July 22nd, 2011 1:00pm.

typed as String properties. As an explanation for the fields, comments were added and exemplary input values specified. In addition, a method was added to the system which determines if only one or two of the three date order positions are set and guesses what the intention of the user was. For example, if position one is set to “YEAR” and position two is set to “MONTH”, position three can only be “DAY”. By default the order is set to year for position 1, month for position 2 and day for position 3, because it is a format often used as a default date format, e.g. by the database management system MySQL[23]. In an evaluation meeting a few weeks after this functionality was developed, it was decided to add two new parameters to the *UserLanguage* class. The parameter *currencyPrecision* of type integer sets the amount of decimal digits for currency values and the parameter *percentagePrecision* which is also typed as an integer value, sets the amount of decimal digits for percentage values. If all values within a column are of the currency or percentage type, the amount of decimal digits is fixed and the contents are aligned to the right of the cell, then the readability is increased.

After the configurator was capable to create new locales, set the parameter values and save the configuration correctly so that Tricia is capable of using it, the *Person*, *Group* and *SystemGroup* assets needed a property to represent the locale settings. In contrast to the system design which suggested a simple association between the assets and the *UserLanguage*, this was not possible, because a field typed with a subclass of the abstract class *Property* was required. Since the locale property can only be of a defined set of values or in other words, a domain, the property was chosen to be a *DomainValueProperty*. Still the domain was missing and therefore the class *LocaleDomain* was added to the *de.infoasset.platform.services.internationalization* package. Finally the *LocaleDomainValueProperty* class was implemented and a *DomainValueProperty* added to *Person*, *Group* and *SystemGroup*. In order to display the added property, no code changes in the presentation layer were needed, because the Tricia platform automatically builds the graphical user interface components and interactions.

Against the methodology described in this Chapter's introduction, the development process followed a traditional development model. At first, the requirements were read and implemented into the system. Afterwards tests were made and the code adapted until the result of this phase was a functioning configuration process for locales and the ability to associate an user and group with a locale.

Phase II – Value, Formatter And Constraint Classes

The core of the implementation associated with this thesis are the value, formatter and constraint classes. In the class diagram depicted in Figure 9 all these classes are related to one type or in the case of *UnitValue* they are related to more than one type. In order to address the part which is critical for success, the test-driven development approach was chosen. Furthermore the development started with a crosscut from left to right in regard to Figure 9. The type chosen for the crosscut was *Date*. At first, the new package *de.infoasset.core.assets.hybrid.typeFormatters* was created and the classes *DateValue*, *DateFormatter* and *DateConstraint* set up with the unimplemented fields and methods from the system design. Then another package called *de.infoasset.hybridWiki.test.hybridPropertyValue* was created with two classes named *DateValueTest* and *DateFormatterTest* inside. The first class contained JUnit[24] test cases for the *DateValue* class and the second one included test cases for the *DateFormatter* class. The tests were run once and, as expected, the tests failed. Gradually the implementation following the test development process improved and subsequently more tests were passed. Some tests had to be

23 MySQL :: MySQL 5.5 Reference Manual :: 11.7 Date and Time Functions, http://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html, Accessed: July 22nd, 2011 1:30pm.

24 Welcome to JUnit.org! | Junit.org, http://www.junit.org/, Accessed: July 22nd, 2011 3:00pm.

corrected, but after the implementation passed all tests, the required functionality was implemented. Internally a date value was represented as a *java.util.Date* object and formatted by the *DateFormatter* using the methods specified in the system design. An internal representation as a *long* value was redundant, because *Date* offered this functionality already. Furthermore the implementation of *DateFormatter* was able to use another Java class called *java.text.SimpleDateFormat*. This simplified the formatting so that only the format patterns had to be created and used by *SimpleDateFormat*. Additionally the *DateConstraint* was implemented without test cases due to the fact that the input from the documentation of the constraint class usage was not precise enough to write any test cases in advance. Still a first implementation was made, but it had to be scrapped after having tested the functionality with the graphical user interface.

Once the approach was found to be working, the other types were implemented, too. As with the date implementation, the structure of *NumberValue*, *NumberFormatter* and *NumberConstraint* was built first. Afterwards tests were written and run to approve their failure. During the implementation process and the ongoing regression tests a problem with the internal value representation of a numeric value as a *double* was found. A test case with an input number with more than five decimal places returned a different value than expected. Due to the limitations of the IEEE standard 754, it is not possible to show the *double* value correctly for large numbers with many decimal places. However, the system requirements did not state anything about the value's precision and since the inaccuracy was merely relevant for non-scientific purposes, it was decided to leave the internal value a *double*, due to the simple programmatic handling and wide serialization support. Still an internal representation as a *java.math.BigDecimal* or even as a *String* is an alternative to the *double* representation. Similar to the *DateFormatter* class which was able to use *java.text.SimpleDateFormat*, the *NumberFormatter* class was able to use *java.text.DecimalFormat*, *java.text.DecimalFormatSymbols* and *java.text.NumberFormat*.

Having tested and completed the number implementation, the idea was to reuse it for the unit value implementation. Instead of creating a pattern that matches a specific unit value, another idea was to separate the numeric part of the unit value from the symbolic part. In order to do this, a method had to be implemented splitting both parts. To write this method was the most difficult part of the *UnitFormatter* class. Since this method was important for the quality of the solution, the tests were written first and then the implementation gradually passed the tests. As a result the *splitParts(...)* method was ready to be used by the *UnitValue* class and its subclasses. The goal of the *UnitValue* class was to reduce code in the subclasses and simplify the addition of more unit based types. As a proof of concept the *PercentageValue* class was implemented with the tests being written first. Except for the *getSymbols() : String []* method and the *isPercentageValue(v : String) : boolean* method all other methods in *PercentageValue* had only one line of code to return a simple value such as the precision. During the implementation of *PercentageValue*, the focus was on how to make the class even more specific, so that *UnitValue* can be used only by setting parameters. Tricia's architecture required that each type had a separate constraint class, therefore only one *UnitValue* constraint was not sufficient. Thus *PercentageConstraint* as well as *CurrencyConstraint* had to be implemented separately. Similar to the *PercentageValue*, the *CurrencyValue* class was implemented with the *getSymbols() : String []* method to return all possible currency names and abbreviations according to the ISO 4217 standard as well as some common currency symbols like “€”, “$” and “£”. Furthermore the *getPrecision() : int* method in *CurrencyValue* was overridden to return the value two and the *isCurrencyValue(v : String) : boolean* method was implemented to return whether it is possible that the given *String* value can be interpreted as a currency value.

As an improvement of the currency type, a parameter in the configuration can be added to set the default currency. Another boolean configuration parameter can determine whether the currency

values will be converted into the default currency as set by the other parameter. In case the boolean parameter is set to true, all foreign currencies will be converted using a conversion rate provider and then presented to the user in the currency s/he is familiar with. A conversion rate provider can be implemented as a web service client for a financial data provider or as a fixed conversion rate configured by an administrator. However this approach was not implemented, because it exceeded the scope of this thesis.

Integration Of The Value-Classes

Since the *DateValue*, *NumberValue*, *PercentageValue* and *CurrencyValue* classes had now been implemented and tested, it was time to integrate them into the platform. At first the occurrences of the already existent classes such as *LinkValue* were examined. As a result the following classes were made out to be changed:

- *de.infoasset.core.handler.Functions*,
- *de.infoasset.hybridWiki.handler.Functions*,
- *de.infoasset.core.assets.hybrid.Hybrid*,
- *de.infoasset.platform.services.asset.change.HybridPropertyChange*,
- *de.infoasset.core.assets.hybrid.HybridSerializable*

In the *Functions* class of the *core* package, only the format for the jQuery calendar had to be set. In order to do so, the *DateValue* class had to be given another static method called *getJQueryUserFormat() : String* which returned the jQuery specific format pattern relative to the user's preferences. Similar to the *Functions* class in the *core* package, the *Functions* class in the *hybridWiki.handler* package mainly had to be adapted for the graphical user interface issues described in the Section below.

The class *de.infoasset.platform.services.asset.change.HybridPropertyChange* had to be adapted in order to recognize changes of typed values. To save changes or new values the serialization process in *de.infoasset.core.assets.hybrid.HybridSerializable* had to be adapted. In addition, the class *Hybrid* in the same package had to be given typing capabilities for the search and sorting function realized with the Lucene[25] engine. Since the sorting algorithm relies on a lexicographic order, each type was given a prefix to determine its rank. Types already present kept their prefix while the new types received their prefix by their degree of detail which was required for the interpretation. Thus *DateValue* was assigned the prefix “a”, *PercentageValue* “b”, *CurrencyValue* “c” and *NumberValue* “d”. For example, in the tabular overview of a type tag, a column contains values of different types. The user requests to sort the values in an ascending or descending order. Subsequently, all rows with date values are shown on top, below the rows containing percentage values, then currency values and finally rows with number values. As this functionality is of importance for various use cases within the system, a test class *de.infoasset.hybridWiki.test.hybridPropertyValue.HybridTest* was implemented to validate the correct behavior.

Integration Of The Formatter-Classes

All formatter classes utilized by the value classes to format the internal values had been automatically integrated due to the fact that no other classes use them. The formatter classes access the locale settings of the system, users and groups.

25 Apache Lucene – Overview, http://lucene.apache.org/java/docs/index.html, Accessed: July 24th, 2011 2:00pm.

Integration Of The Constraint-Classes

In contrast to the generally difficult integration process of the value and formatter classes, due to many code refactorings by other developers and missing documentation, the integration of the constraint classes was straight forward. At first, the new constraint classes were added to the list of constraints in *de.infoasset.core.assets.hybrid.typeConstraints.AbstractTypeConstraint* which also made the new types visible for the user. Then the constraints had to be added to the instantiation process of new constraints in *de.infoasset.core.assets.hybrid.HybridPropertyDefinition*. Furthermore a check in *de.infoasset.hybridWiki.handler.Functions* had to be added in order to initialize the dialog to show the correct type, if an attribute definition is set. Finally the additional constraints were tested using blackbox tests and evaluating each type as a constraint in an attribute definition.

Phase III – Graphical User Interface

To sum up the integration of the solution's core, it was possible to merge all value, formatter and constraint classes into the system to continue adapting the graphical user interface. While the Tricia platform contains both a client and a server platform, the server part is based on Java and the client platform on a combination of three technologies mainly for the presentation layer: HTML, Cascading Style Sheet (CSS) and Javascript. The latter is used in conjunction with the jQuery framework and its extensions for graphical user interface enhancement called jQuery-UI and a plug-in for Tables named DataTables. User requests are answered by the server with formatted responses which are formulated in HTML and CSS and stored in templates on the server. The mechanism to embedded data in the HTML/CSS response uses substitution. For example, the word “$value$” can be contained in an HTML/CSS code and will be replaced by the platform with the correct substitution value before it is sent to the client.

From a technical perspective the attribute value is substituted for a placeholder within a HTML/CSS document. The dialog to add and edit an attribute as shown in figure 12 on page 44 is a template and therefore it had to be adapted in order to accommodate the type selection menu as depicted above. These changes affected all occurrences of the dialog, because the dialog is always created on the basis of the template and thus it was possible to implement the functional consistency demanded by the literature. Furthermore the interaction between the input field, the calendar icon which adds a calendar to the input field and the type selection menu, had to be programmed. Since the selection menu was necessary for the interactions, it was added to the attribute dialog template as another template containing a HTML *select* element named *typeSelection* and its option elements which comprised the name of the types. Afterwards the *hybridWiki.js* file was adapted to show the calendar icon if the selection menu item *Date* was selected or the user created a new attribute value with the type *Text* preset. In case the user clicks on the calendar icon, a calendar is shown attached to the input field. In case another type such as *Number* is chosen the calendar icon will disappear and the attached calendar will be removed from the input field. On form submission, the value of the selection menu is read and processed as depicted above. Except for these changes using Javascript and CSS, all other changes were made in the Java code of the server application. Still this implementation was complex, because all other interactions and mechanisms already in place must be kept unchanged, although the Javascript code was used in different contexts. Thus the debugging process was complicated and a separate debugging tool called Firebug[26] had to be utilized.

The detailed result of the graphical user interface adaption is shown in the Figures and paragraphs below. As mentioned in the paragraph above, the dialogs are built from the templates and thus only one example per template was chosen to be depicted in detail.

26 Firebug, http://getfirebug.com/, Accessed: July 24th, 2011 4:00pm.

Attribute Dialog

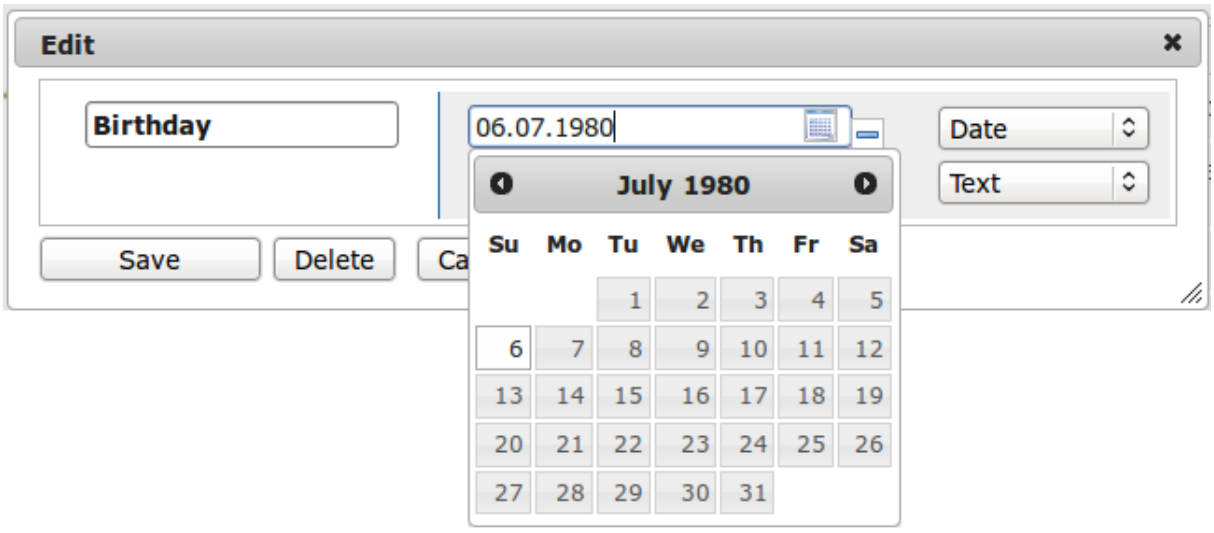


Figure 19: Changed Attribute Dialog
[Source: Exemplary screenshot of Tricia taken by the author]

In Figure 19 the editing dialog of an attribute is shown. In contrast to Figure 12 the dialog was altered roughly as Figure 13 shows. For space reasons however, the calendar icon was moved from behind the minus-icon to the end of the input field. This added two major advantages: first, the calendar icon is visually related to the input field where the value from the calendar is inserted and second, the calendar opens below the calendar icon instead of next to it. The length only exceeds the space in front of the calendar icon, if the value of the second's fraction is longer than 4 digits. However, since a maximum of three millisecond digits are used to generally display the fraction of a second, it was decided the space was wide enough.

Formatted Attributes

Figure 20: Formatted Attributes
[Source: Exemplary screenshot of Tricia taken by the author]

Figure 20 is an extraction of a wiki page showing the associated attributes formatted according to a German locale with the thousand separator as a dot, the decimal separator as a comma and the date order as day, month, year – separated by a dot as well. In case a number is inserted in the input field and the type currency is manually chosen, the value is saved as text and not formatted. This is due to the missing configuration parameter for the default currency and its convertibility as explained on page 51.

Locale Preferences

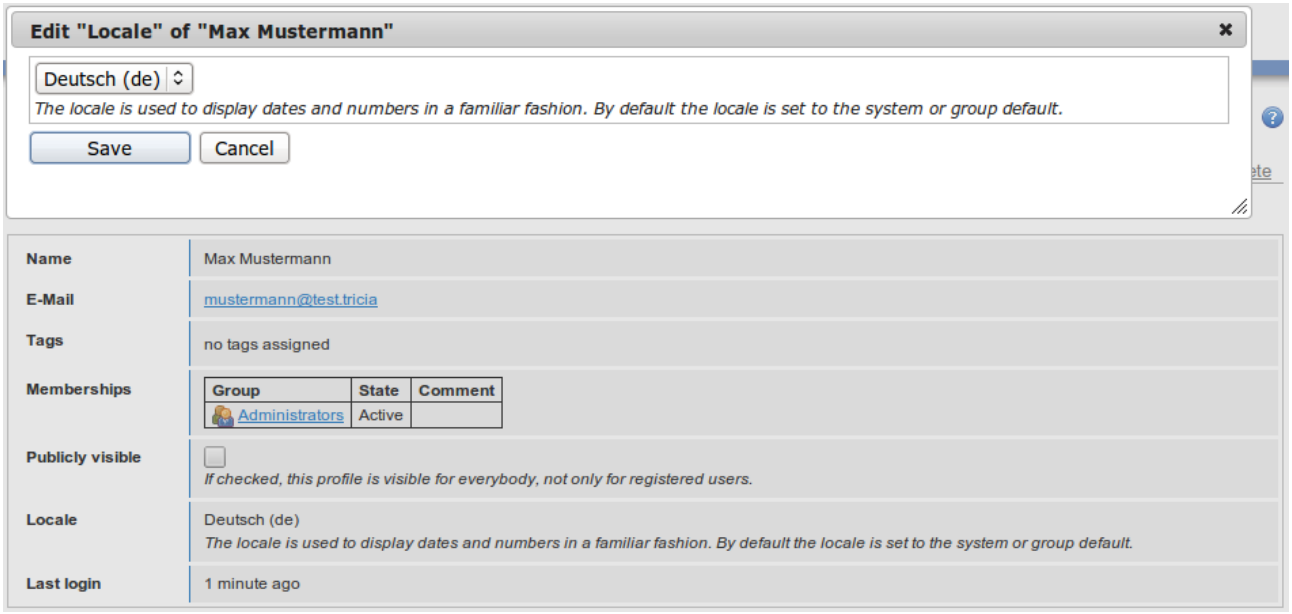


Figure 21: User Profile With Locale Setting
[Source: Exemplary screenshot of Tricia taken by the author]

The user configuration is shown in Figure 21. An overlay of the configuration page shows the locale editing dialog. The locale currently set is shown as the selected item in the selection menu, although this can be changed by simply selecting another item from the menu. Underneath the overlay, the page is displayed with a gray shadow. As the second item from bottom, the locale parameter is shown as it is displayed in the selection menu. Finally the additional parameters were integrated in the graphical user interface as planned and depicted in context of Figure 17.

Type-sensitive Sorting

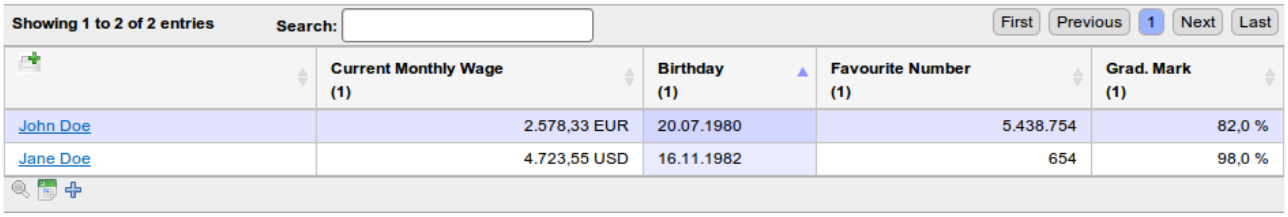


Figure 22: Sorted Tabular Type Tag View
[Source: Exemplary screenshot of Tricia taken by the author]

All pages associated with a type tag are displayed in the type tag's overview as shown in Figure 22. In contrast to Figure 16, the values are formatted according to the user's locale, respectively in the same way as shown in Figure 20. In order to present the new sorting capabilities, the birthday of "John Doe" was changed to July 20th, 1980 instead of July 6th, 1980 as in the previous Figures 19 and 20. Thus sorting the birthday column in an ascending order used to result in "16.11.1982" first and then "20.07.1980", because the first character of the string was a "1" in the first case and a "2" in the second case, although 1982 is after 1980. Now, as shown in Figure 22, the birthday column is sorted correctly. This was achieved, because the sorting algorithm receives an internally formatted representation of the attribute value instead of its displayed representation. The displayed representation of a number, percentage and currency value is aligned to the right while all differently typed values are aligned to the left of a cell. In case a column contains values of different

types, the priority of the type determines the values rank. As described in the Chapter “Integration Of The Value-Classes”, a date value has the highest priority due to its most specific format, then follows the percentage, currency and number values concluded by the link and text values.

Outcome

Concluding the implementation phases, the overall result is similar to the planned product. Only a few divergences such as a more complex way to associate an user with a locale were made. Still, the implementation has room for improvement like a more refined way of differentiating between types and a value preview for the attribute dialog.

In detail, the configurator was adapted as planned, although the locales defined in the configuration had to be associated with an user, group or system group by writing two new classes called *LocaleDomain* and *DomainValueProperty*. Furthermore, the order of a date was implemented as three different *String* values instead of an enumerated array parameter. Still, the configurator as well as the configuration interface for an user, group and system group were adapted as planned.

The concept of test-driven development improved the quality of the value and formatter classes. As displayed in the following Figure, nine classes with over 30 test cases and numerous test values resulted from the testing activities.

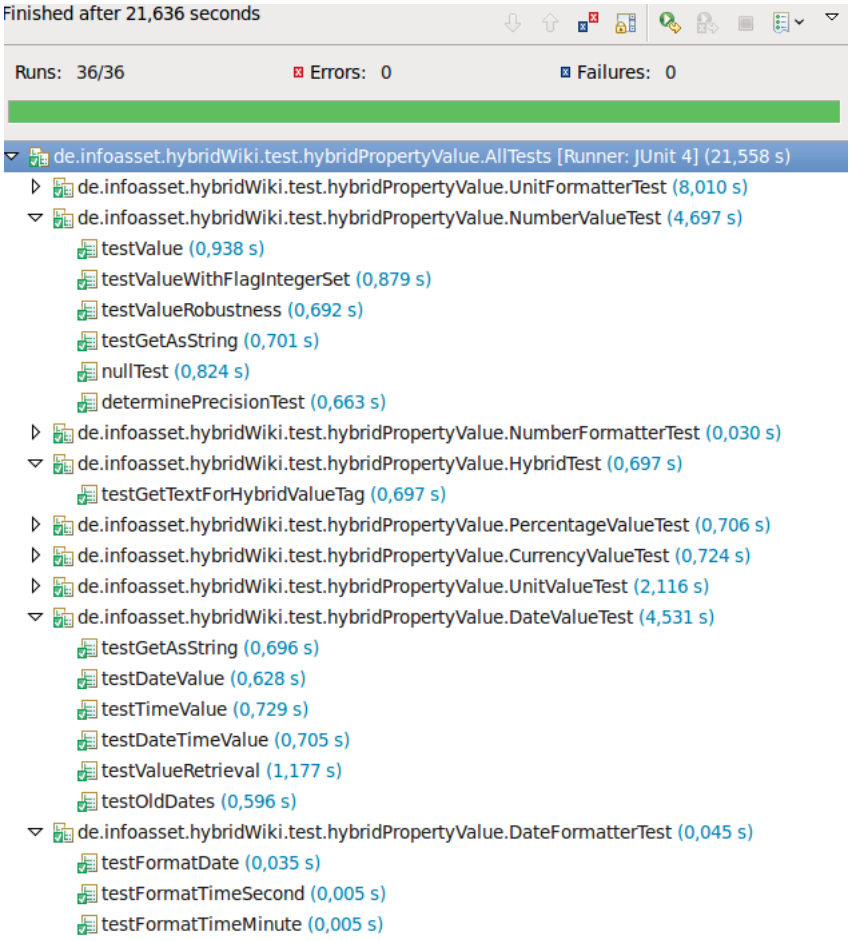


Figure 23: Test Results
[Source: Screenshot of the Junit 4 results as displayed in Eclipse]

The system layout constructed during the system design phase was implemented with minor changes such as adding some private methods for convenience and readability. During the realization of the system design, especially the *UnitValue* class showed itself to be very extensible. New types like weight, size or volume can be added to the system quickly, but the integration into Tricia will require more in-depth knowledge about the system's architecture and typing system. Since the value types build on the formatter classes, these classes offer formatting capabilities which can also be used by other parts of Tricia. For example, the conversion of dates from any locale to a database management specific format can be performed by the *DateFormatter* class in case a locale with the correct settings is provided.

Since the graphical user interface is the part of a system which is exposed to the user, it is this part which determines the user-friendliness. The changes proposed during the system design phase considered the specific requirements and thus it was essential to implement the changes precisely. Adding and editing an attribute works as planned on a wiki page as well as in the tabular overview of a type tag. Furthermore an attribute is displayed with the formatted values on a wiki page and in a type tag's tabular overview. If an attribute definition for an attribute exists and a type constraint is set, the type is predefined while adding a new attribute value. The sorting function was also changed so that the type specific order of the values was regarded.

An overview of the different representations of an attribute value is presented in Figure 24. The user enters a value (*Input by User*), for example "-1488.579€". This value is represented within the system as the numerical value -1488.579 and its currency symbol "€" (*Canonical Representation*). In order to sort the value along with other currency values, another representation has to be offered so that the sorting engine is capable to order the values with a lexicographic sorting algorithm. However, neither the internal representation nor the *Sort Value* representation can be used to store and transfer the attribute value and thus the a representation (*JSON*) for the JSON[27] technology is needed. Since the *JSON* representation is used to store and retrieve the value from the storage, its graphical representation box in Figure 24 stretches from the *Input* box to the *System Internal* box. Finally the value has to be presented to the user in different formats, depending on the context and locale of the user's account. The representation (*Edit Attribute*) of the value in an edit dialog has to be as precise as the user has inserted the value into the system, but the value displayed on a wiki page (*Formatted Attribute*) or in the tabular overview of wiki pages associated with a type tag (*Formatted Attribute in Table*) for example, only need to show as many decimal digits as specified in the configuration. To serve all components with the correct representation of a value, a strong formatter class was developed and tested thoroughly as depicted in Figure 23 above.

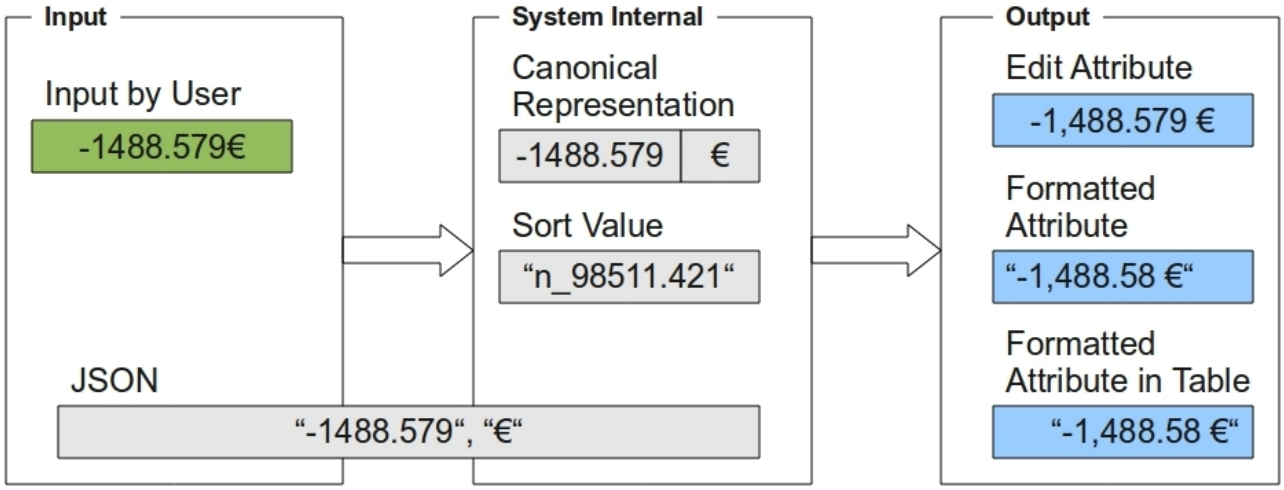


Figure 24: Representations of a Currency Value [Source: Author's design]

27 JSON, http://www.json.org/, Accessed: August 5th, 2011 6.00pm.

In addition to the summary of the efforts needed to implemented the presented features, the obstacles are depicted in this paragraph. At first, the implementation went smoothly until the derivatives of *HybridPropertyValue* had to be integrated into the platform. It was hard to read undocumented code in order to understand the procedures necessary to integrate the classes. For example, a similar serialization mechanism had to be implemented three times, although it was not documented for the developer why this was necessary. In addition, the constantly changing code basis made it even more difficult to test the integrated parts, but if the code was written against a stable basis, the integration at the end of the project would have been even more complicated. Lastly the complicated technology stack, used to dynamically generate the client interface, made the GUI development faulty and tedious. Especially the use of the same Javascript code in different contexts and the need for another debugging tool prolonged the development process. However, the produced and integrated code in the context of this thesis has Javadoc[28] comments for every method, field and parameter, even in the test classes and in the Javascript resources. Since a developer is not supposed to be involved in other test stages than the unit tests [ISTQB 2011, p. 47], the integration, system and acceptance tests will most likely depict more bugs which can be fixed, because of the straight forward design and due to a thorough documentation.

28 Javadoc Tool Home Page, http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html, Accessed: July 25th, 2011 6:00pm.

Conclusion

In the last Chapter of this thesis, the conclusion from the research, system design and development process is drawn and an outlook is given. Afterwards the work is put in perspective.

Learnings from the Research

During the research phase, the literature was searched for types, formats and constraints first. A collection of types was compiled containing both practically relevant types such as the basic primitive types, enumeration, sequence, reference, etc. and rather theoretical types like the null type and placeholder type. Exemplarily the date type was discussed in detail. In addition, the literature was searched for principals in interactive management. As a result three main aspects have to be considered when designing a graphical user interface: the human factors, the application's presentation and the interaction between the user and the system. These factors were the criteria for the GUI design later in the process.

After searching the literature, software tools which had already solved similar problems were examined. At first, Microsoft's Office 2010 products Excel and Access offered similar capabilities to structure data using Tables. Both tools included similar types, formats and the opportunity to constrain values. However, Access's type enforcement was stricter than Excel's, especially considering that the user has to decide on the type of the data before entering a value. Another major difference between both tools was the ability to import and export data. While Access includes multiple options to integrate data sources, Excel is far more limited. Excel however, offers better capabilities to format data such as a different formatting of the same value type in the one column. As a free alternative to Microsoft's Office suite, OpenOffice.org was examined, too. In general the graphical user interface looks similar to old versions of Microsoft Office and the difference is only visible in detail. Formats in OpenOffice.org are indexed and attached to containers like pages, Tables or paragraphs, but the variety of types and formats offered by OpenOffice.org Spreadsheet are the same as in Microsoft Excel. Similar to Microsoft Access, all values in SAP ERP are typed and the formats are being defined during the customization process. The types offered by SAP ERP are similar to the SQL standard of 1992 and a subset of the types Microsoft Access offers. As in SAP ERP, types can be defined programmatically also in SugarCRM which is a Web 2.0 business software. Contrary to SAP ERP however, SugarCRM does not allow the user to specify a type for the input data. All types are defined during the application's development, but since the application is a open source product, an user with programming skills is able to change data types. Furthermore SugarCRM allows the user to choose a locale for an user account which all numbers, dates, times and currencies are formatted with. The software tools mentioned above were chosen, because of their dissemination and popularity. Further research on the use of types, formats and constraints can be done examining other Wiki systems and Web 2.0 tools.

As a result of the findings from the literature and the software tools examination, a list of common types was created and regarded as the basis of all further considerations. Except for the binary type whose implementation would have exceeded the scope of this thesis, all other types were implemented. Furthermore the saving and formatting mechanism found in all evaluated software systems was similar: At first the user supplies a value which is evaluated and prepared to be stored in a database or something alike, then the value is fetched from the database, formatted and written onto the screen. Since Tricia had a mechanism compliant to the one described, it was adapted to match this process. However, since the constraint mechanism used with the attribute definitions differs widely from the mechanisms used in the evaluated software tools, the implementation present in Tricia was extended with the newly implemented types.

Evaluation of the System Design

The mechanism used by the examined software tools to save and format values, was the basis for the processes associated with the typing mechanism. Furthermore the list which resulted from the initial research phase was taken to determine which values had to be added to Tricia in order to support all common types. In addition, the conceptual design had to be mapped to Tricia and thus an insight view of the system and technologies used was necessary.

Before the system design was approached, the requirements and use cases relevant for this thesis were set. As a non-functional requirement high usability as described in the literature had to be strived for as well as high maintainability since the produced code was an extension of the core functionality and it had to be integrated into a project with multiple developers. Functional requirements were to add the types date, number, currency and percentage to the system, to empower the user to change the type of an attribute and to enable the sorting function to handle type-sensitive values. With these requirements set, the specific use cases had to be appointed in order to set the affected parts of the platform and to get a criteria for project completion. The major use cases were to set a locale for each user, group and system group as well as for the whole system applying the configurator. Furthermore the user has to be able to view an attribute with a formatted value, change the type of a value and save the value with the changed type. Furthermore the enforcement of type and multiplicity constraints had to apply to the newly added types, too. Additionally the user had to be able to view a formatted value in the tabular overview of wiki pages associated with a type tag.

Now that the requirements and use cases were defined, the mechanism to save a typed value and retrieve it from the storage had to be described in detail. Simply put, the process starts by evaluating the user input and the recognition of the value's type. The recognized type is shown to the user and s/he is given the opportunity to change the type. On saving the attribute, constraints are applied and eventually an error or warning message is displayed. Finally the value is stored and displayed if the process has succeeded. This procedure corresponds to the mechanism observed in the examined software tools. The objects needed to perform the tasks required by the described procedure as well as the formatting parts had to be structured. Thus a conceptual system design was created with three main parts: the value types, validators and formatters and one peripheral part: the configuration. The value type classes represent a typed attribute value, the validators represent the constraints and the formatter classes purpose is to recognize the format of a value and format it according to the configuration which is represented by the locale class. Except for this clear separation, the design contained a generalization of all unit value types like percentage or currency. The abstract unit value type utilizes the number type and its formatter class and defines the methods and fields common to all unit value derivatives. Still, the unit value class has no validator, due to the fact that only specific values can be evaluated and thus only specific type constraints are required. Finally this approach was mapped to Tricia. As a result the design varied mainly by using different class names and adding methods to each class. Only the internationalization package had to be mapped to the chosen *Principal* derivatives and decided to change the already present *UserLanguage* class. One thing that has not been regarded in the system design phase, was the sorting algorithm of the tabular overview of wiki pages associated with a type tag. Since the evaluation of the platform concluded that only minor changes had to be made, it was decided to leave this issue for the implementation phase to solve.

Although the system design had numerous benefits, the amount of classes could have been reduced by merging the formatter class into the value type classes. Furthermore the locale settings could have been applied to the *Principal* class instead of each subclass in favor of a more simple internationalization package. For implementation and maintainability reasons however, the

presented design was chosen, because smaller classes and the logical separation of value types and formatters offered a higher maintainability, better reusability, especially in regard to the formatter classes, and a better extensibility. The latter became apparent in terms of the unit value hierarchy.

Still, the best system design is not much worth if the users are unsatisfied with the product. Therefore the GUI changes had to be planned precisely and carefully. The most important changes were to add a selection menu for the type of the attribute value. It shows the text type by default, but alters its selection when another type is recognized. In case the dialog has just been initialized, the calendar icon is shown to offer a quick access to enter a date, but as soon as the type is changed to another type but date, the calendar icon is hidden. To maintain a consistent functionality this behavior was applied to all occurrences of adding and editing an attribute. Also all occurrences of the attribute value's representation had to be changed to show the formatted value. Furthermore the attribute definition dialog received new entries for the added types while the new types still had to offer the same functionality in regard to the level of enforcement as the ones already in Tricia. Additionally the configurator had to be adapted in order to contain the configurable parameters for locales. Overall the graphical user interface offers a consistent functionality, an improved aesthetic appeal and speaking descriptions. However, these improvements can be validated by an user study in the future.

Concluding the system design, the basic saving and formatting mechanism supplied a solid foundation from which the system design originated. The system design itself had a clear separation to offer high extensibility and maintainability. Finally the graphical user interface changes were planned in detail and considered the recommendations found in the literature.

Evaluation of the Implementation Process

On the basis of the system design phase, the implementation phase started adapting the internationalization package and the configurator. Afterwards the value, formatter and constraint classes were implemented and integrated into the platform. During this phase the lack of code documentation as well as the constantly changing code basis made it difficult to integrate the newly added classes. Despite the obstacles, the integration was successful and all test cases passed. Finally the graphical user interface had to be adapted as planned. However, this was rather difficult due to the use of Javascript in various files and different page contexts. Thus it was not possible to completely adapt all occurrences of the attribute adding/editing dialog. To improve the maintainability of the typing system, all classes, methods and parameters in the scope of this thesis were annotated with Javadoc comments. Furthermore the implementation was kept as simple as possible to offer high maintainability and especially the formatter classes which can be utilized by other parts of the platform, add reusability.

In the end, the improvements implemented within the scope of this project enable the user to control the types of an attribute value, to view all attribute values in a format compliant to his or her locale and it enables the user to sort values type-sensitively. Within the platform, the added typing mechanism empowers the structured query implementation to compare attribute values based on their type and to sort the results.

Since no software is free of bugs and functions can always be added to a software, there is enough room for improvements. Additionally, the current version of the implementation has only been tested by the developer, further tests are likely to show more bugs. A known issue is the interpretation of values such as “01.10” with the thousand separator set to “.”. In this case the value is interpreted as the number “110” instead of the date “January 2010”. This examples shows that the

implementation of the type recognition algorithm has to be refined for special cases and due to the complexity and the lack of performance, the process can be parallelized. Furthermore the user input value can be saved separately from the interpreted and internally stored value in order to guarantee the initial value is not lost by accidentally changing the value to an incompatible type. This issue can also be prevented by showing a value-preview next to the selection menu in the adding/editing dialog of an attribute. Thus the user would have an immediate result of his or her type choice and s/he would be able to change the type if the value is not in the user's intention. Another issue is the order of types used by the sorting algorithm of the tabular overview of wiki pages associated with a type tag. In case the column which the results are sorted by has a type constraint, the values are presented in the type-order defined by the developer. A better solution is to show rows with values matching the type constraint on top and all other values below. Finally more types can be added to the typing system, especially the binary type to support media content which has become an essential part of the web during the last few years.

Work in Perspective

This thesis concentrates on the typing aspect of attribute values in a Hybrid Wiki. The Hybrid Wiki is a lightweight concept to structure information and was lately described by the paper Hybrid Wikis: Empowering Users To Collaboratively Structure Information [Matthes/Neubert/Steinhoff 2011] which was presented on July 18th, 2011 at the 6th Conference on Software and Data Technologies in Seville, Spain (ICSOFT 2011). The Hybrid Wiki implementation this thesis is based on is integrated in the Tricia platform, an enterprise collaboration and information management platform.

From the individual perspective of an administrator the implemented changes in Tricia result in more work during configuration. The locales have to be configured and the users, groups and system groups can be associated with locales. This prolongs the configuration process, but weighted with the increase of productivity for the enterprise, the additional costs for configuration are tolerable. From the perspective of a software developer who has to maintain the platform additional code is always more work. However, the additional code is well documented, extensible and reusable and thus the improvements overweight the drawbacks. From an user perspective, s/he is empowered to control the value types, view formatted values and sort type-sensitive. This increases the productivity of the user, because it reduces misinterpretations of numbers and dates and increases the precision of search results. From a business perspective the implemented features reduce the costs for information management and thus justify the additional costs for administration and implementation. From a scientific perspective a mechanism to save, constrain and format values was identified and described and a corresponding architecture created and verified by an exemplary implementation. In addition, the thesis contains a list of types and a selection of the most important types which can be used by other projects who have to solve similar problems in the future.

Appendix A – Detailed List of Data Types

Primitive Data Types

Boolean

The type Boolean is to distinguish between two values: true and false.

Possible Values	true, false
Formats	0 / 1 , True / False
Order	no default order
Constraints	Depends on the input, for example, if the input is a text field it should only allow the *words* named above (formats)
Saving	can be saved in one bit as 0 or 1
String Representation Examples	Yes / No, T / F, True / False 0 / 1

Table 20: Type Details - Boolean

Representation Examples

Figure 25: Boolean - Representation Examples

Numbers

Integer Numbers (Integer and Long)

Integer and Long numbers are types to represent discrete values.

Possible Values	depends on the system's memory and the implementation of the type mathematically: $-\infty$ to 0 to $+\infty$ programmatically: a finite subset of the mathematical definition often $-(2^{15})$ to $2^{15}-1$ for the Integer type and $-(2^{31})$ to $2^{31}-1$ for the Long type
Formats as regular expressions	Signed number [-+]?\b\d+\b Scientific notation ((\b[0-9]+)?\.)?\b[0-9]+([eE][-+]?[0-9]+)?\b with or without a thousand-separator such as ' or . or ,
Order	-n, -(n+1), … , -1, 0, 1, …, (n-1), n ; where -1 < 0 < 1 is true
Constraints	*See regular expressions above*
Saving	Can be saved as a bit string of usually 8 or more bits
String Representation Examples	123 2^8 $3.4 \cdot 10^2$ 54'012'987,56

Table 21: Type Details - Integer and Long

Representation Examples

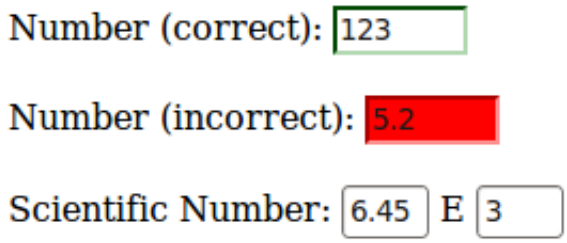


Figure 26: Integer/Long - Representation Examples 1

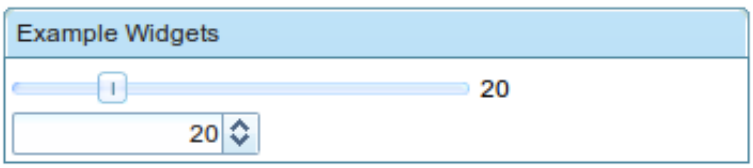


Figure 27: Integer/Long - Representation Examples 2

Floating Point Numbers (Float and Double)

The floating point number types Float and Double represent decimal values.

Possible Values	depends on the system's memory and the implementation of the type mathematically: -∞ to 0 to +∞ programmatically defined in the standard IEEE 754-2008
Formats as regular expressions	Signed number (\b[0-9]+\.([0-9]+\b)?|\.[0-9]+\b) Scientific notation ((\b[0-9]+)?\.)?\b[0-9]+([eE][-+]?[0-9]+)?\b with or without a thousand-separator such as ' or . or ,
Order	-n, ..., 0, ..., n ; where -n < 0 < n is true
Constraints	*See regular expressions above*
Saving	Can be saved in 16, 32, 64 or more bits.
String Representation Examples	1,2 1.2e34 $1.2345 \cdot 10^3$ 12,345.67

Table 22: Type Details - Floating Point Numbers

Example Representation

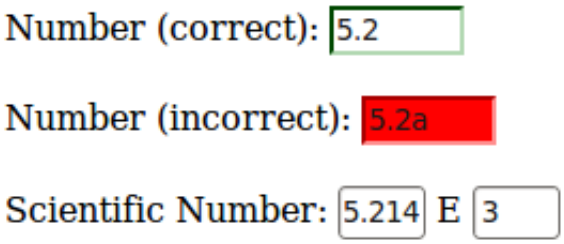


Figure 28: Floating Point Numbers - Representation Examples

Other Numbers

In addition to integer and floating-point numbers there are several other number types such as fractions, imaginary numbers, not ending numbers (for example pi) that cannot be displayed as bit strings. Complex types or the text type is used to represent such values. A more detailed view on the topic is not part of this thesis.

Strings

Text

A text string is any type of character array where the characters can be either alphanumeric characters or symbols.

Possible Values	alphanumeric characters, symbols
Formats	Left-to-right, right-to-left, different fonts, colors, bold, italic, lined text, headings, quotations, character sets, etc.
Order	Lexicographical, string length, capital → small, semantic order, etc.
Constraints	only encoded characters supported by the host system
Saving	Character array where each character represents a numerical value which can be decoded as a binary number and stored in bits.
String Representation Examples	Hello World.)!§“%K-²]³[}M_NC/!“$MCafh9213'*Ä!!()%/§

Table 23: Type Details - Text

Example Representation

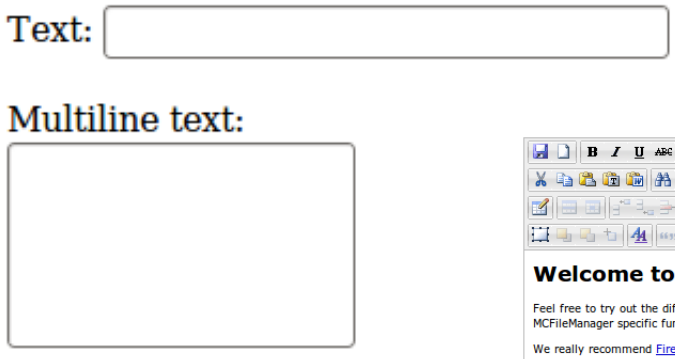


Figure 29: Text – Representation Examples 1

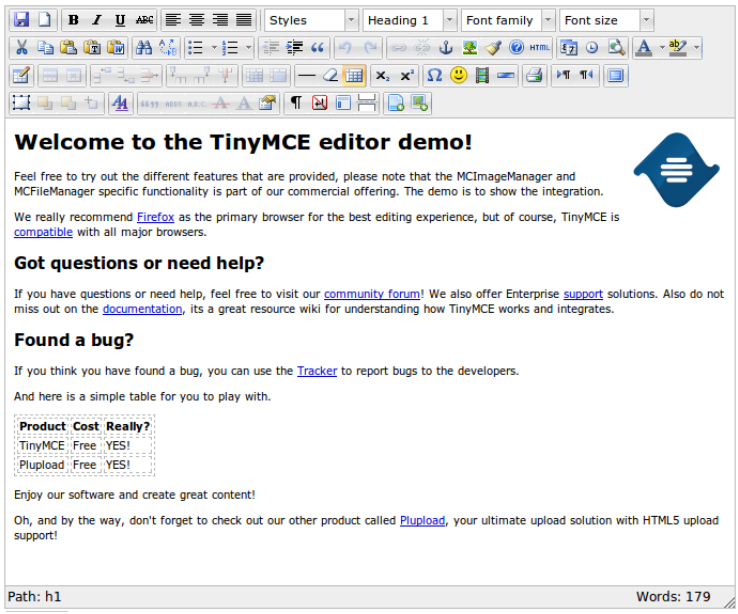


Figure 30: Text - Representation Examples 2[29]

29 This graphic is an extraction of a screenshot from the following page: http://tinymce.moxiecode.com/tryit/full.php, Accessed: July 27th, 2011 5:00pm.

Date, Time

A string to represent a date, a time or a combination of both is a common representation for time for most computer systems.

Possible Values	Date: January 1st – December 31st, year 0 to ∞ Time: Hour 0, Minute 0, Second 0, Second Fraction 0 to Hour 23, Minute 59, Second 59, Second Fraction < 1
Formats	Common date formats: <day>.<month>.<year> <year>-<month>-<day> <month>/<day>/<year> Common time formats: <hour>:<minute>:<second>,<decimal fraction> <hour>.<minute> Combined formats[30]: YYYY-MM-DDThh:mm:ss.sTZD (eg 1997-07-16T19:20:30.45+01:00)
Order	Date: January 1st is before January 2nd Time: 00:00 is before 00:01
Constraints	only possible dates and times (see possible values), input constraints depend on input method
Saving	• as an array of integers • as a double value of a certain date, e.g. 1900-01-01 as in Microsoft Excel/OpenOffice.org Calc ○ calculations partly with negative numbers
String Representation Examples	1994-11-05T08:15:30-05:00 corresponds to November 5, 1994, 8:15:30 am, US Eastern Standard Time

Table 24: Type Details - Date, Time

Representation Examples

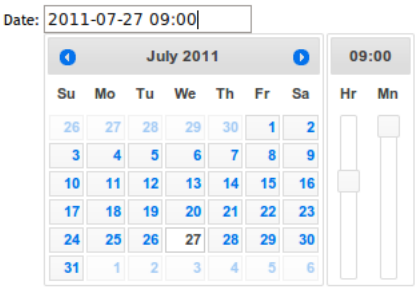


Figure 31: Date, Time - Representation Example 1[31]

Figure 32: Date, Time - Representation Example 2

30 Date and Time Formats, http://www.w3.org/TR/NOTE-datetime.html, Accessed: July 11th, 2011 12:00pm.

31 This graphic is an extraction of a screenshot from the following page: http://jquery.developmental.co.za/datetime, Accessed: July 27th, 2011 5:15pm.

Currency

The currency type usually represents amounts of money in a certain currency.

Possible Values	The same as a floating point number plus a value for a currency.
Formats	A decimal number with a predefined amount of significant figures, usually two, and a symbol, short form or full name of a currency
Order	0,00 <currency> is less than 0,01 <currency>
Constraints	Currencies as in ISO 4217 (for current use) or currencies for
Saving	as two values, one a floating point number and the other a reference key to a enumeration of currencies
String Representation Examples	12.34 € 567.89 USD 0.459 British Pound

Table 25: Type Details - Currency

Representation Examples

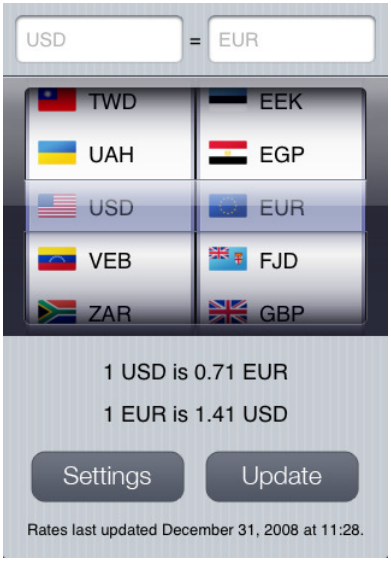


Figure 34: Currency - Representation Example 1[32]

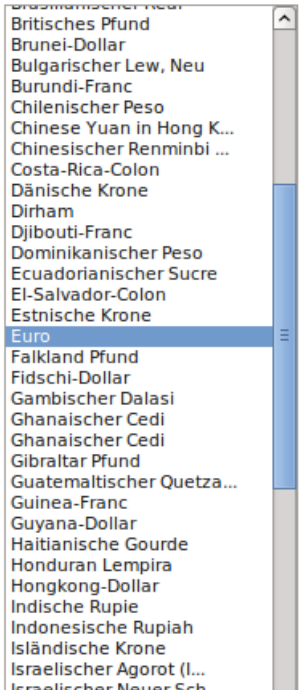


Figure 33: Currency - Representation Example 2

Comment

Any other unit can be used similar to currency. Other units are for example: percentage, meter, hectare, liter, pieces, pascal, Celsius, et cetera. It depends on the locale whether the unit is in front or after the number.

32 This graphic is copied from the following page: http://touchreviews.net/wp-content/uploads/2009/05/touch-reviews-iphone-apps-curcon-currency-converter_1.jpg, Accessed: July 27th, 6:00pm.

Error

An error is a string representing a misbehavior of a part of the system including the user.

Possible Values	any text
Formats	no standard format
Order	no order
Constraints	no constraints
Saving	Often errors are just displayed, not saved. If they are saved they are often stored in log files or bug repositories.
String Representation Examples	HTTP 404: File Not Found. Permission Denied. NullPointerException Kernel Panic

Table 26: Type Details - Error

Representation Examples

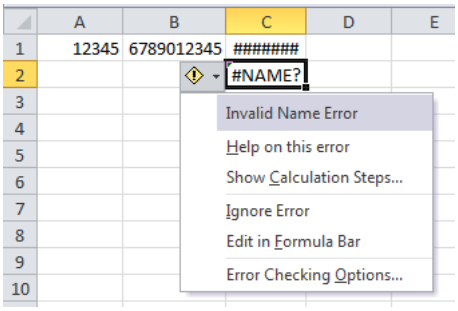


Figure 35: Error – Representation Example 1

Figure 36: Error – Representation Example 2

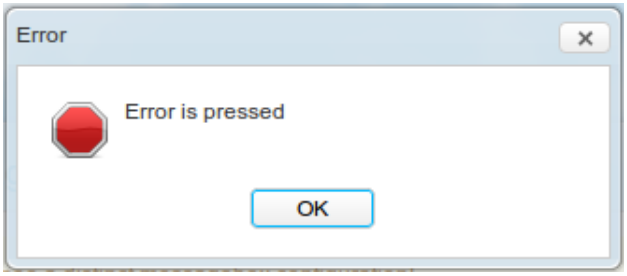


Figure 37: Error - Representation Example 3[33]

33 This graphic is an extraction of a screenshot from the following page: http://www.zkoss.org/zkdemo/window/message_box, Accessed: July 27th, 2011 6:15pm.

Enumeration

A sequence of indexed values is an enumeration.

Possible Values	A finite sequence of equally typed objects.
Formats	list of values
Order	The order is defined by the one who defines the sequence.
Constraints	Only objects of the same type can be in an enumeration.
Saving	The internal representation depends on the implementation. Can be stored in a hash Table.
Examples	[Apples, Plums, Bananas]

Table 27: Type Details - Enumeration

Representation Examples

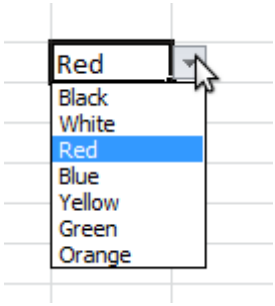


Figure 38: Enumeration - Representation Example 1

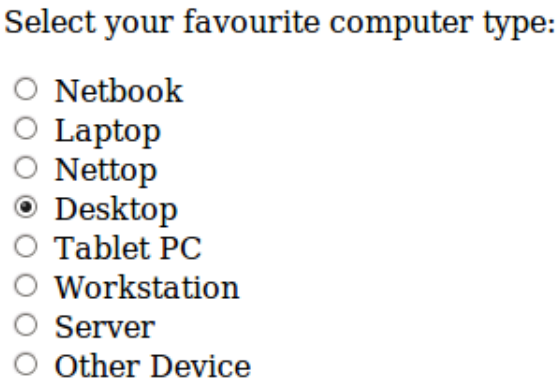


Figure 39: Enumeration - Representation Example 2

Binary Data

Image

An image is a set of pixels or vectors and colors.

Possible Values	Pixels, vectors, colors in combination
Formats	Multiple file formats
Order	Images can be ordered with their *filename, file size and image size* apart from semantic-based orders
Constraints	The amount of available colors is limited to the software and hardware compatibilities of the system that displays the image.
Saving	Usually saved in a file.

Table 28: Type Details - Image

Representation Examples

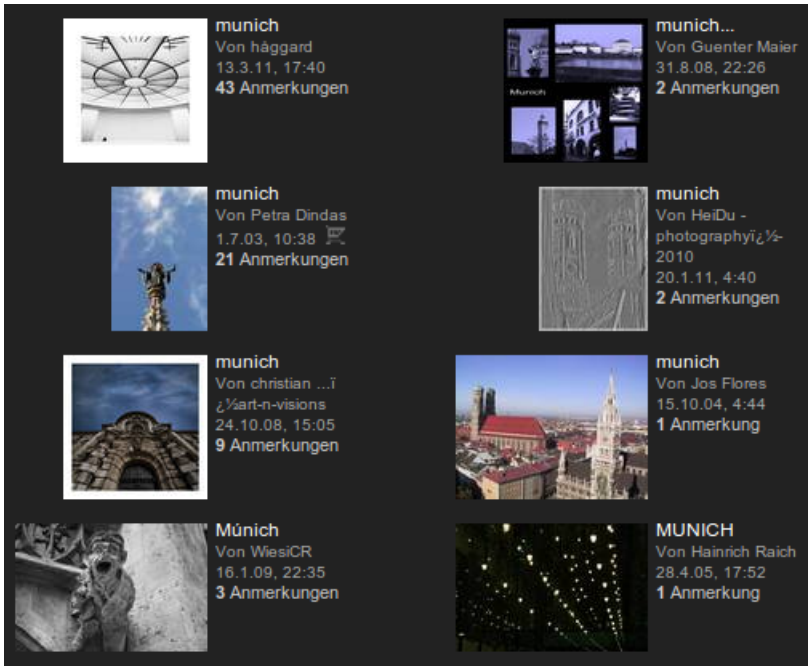


Figure 40: Image - Representation Example[34]

34 This graphic is an extraction of a screenshot from the following page: http://www.fotocommunity.de/search?q=Munich&channel=3, Accessed: July 27th, 2011 6:30pm.

Audio

The audio types consists of sound waves being saved with certain levels and usually compressed.

Possible Values	Wave definitions and their levels.
Formats	Multiple file formats
Order	Audio files can be ordered with their *filename, file size and image size* apart from semantic-based orders
Constraints	The quality is reduced when compressing the audio inputs. Furthermore the sound quality depends on the microphones and pickups recording the sound.
Saving	Usually saved in a binary file.

Table 29: Type Details - Audio

Representation Examples

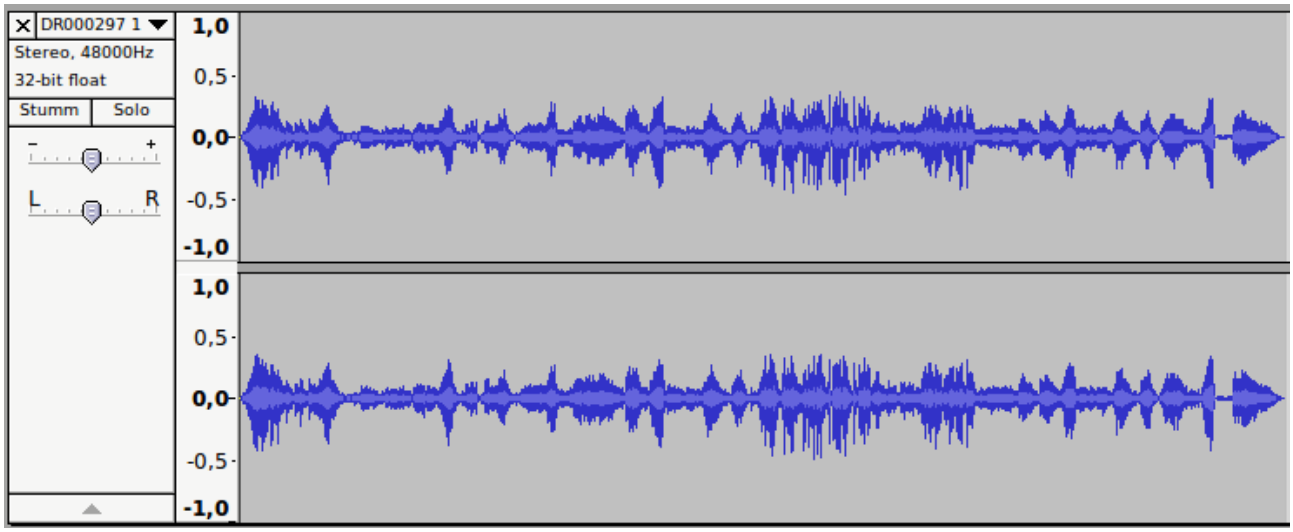


Figure 41: Audio - Representation Example 1

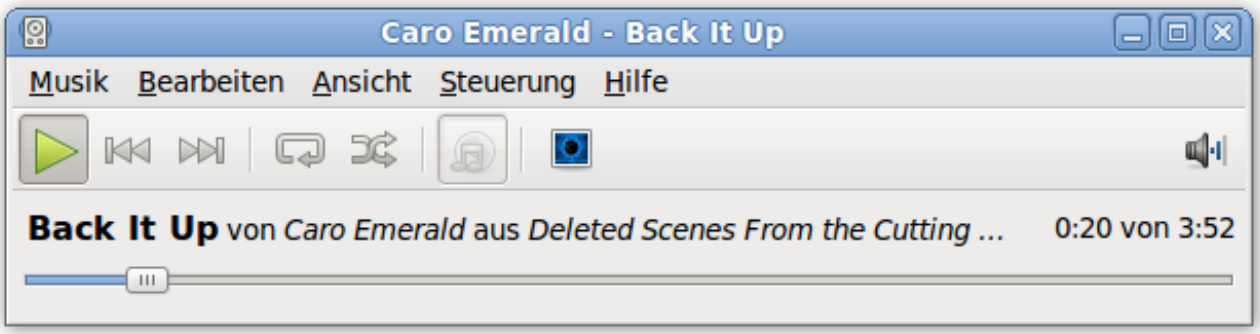


Figure 42: Audio - Representation Example 2

Video

A video is a combined type of images in a time context together with an audio track.

Possible Values	combinations of images and sound
Formats	Multiple file formats
Order	Video files can be ordered with their *filename, file size and screen size* apart from semantic-based orders
Constraints	The quality is reduced when compressing the video content.
Saving	Usually saved in a binary file.

Table 30: Type Details - Video

Representation Examples

Figure 43: Video - Representation Example[35]

35 This graphic is an extraction of a screenshot from the following page: http://www.in.tum.de/en.html, Accessed: July 27th, 2011 6:30pm.

Other File formats

Other binary data is usually stored in files.

Possible Values	Any type of binary data in a file format
Formats	Any file format
Order	Video files can be ordered with their *filename, file size*
Constraints	System storage is not big enough to store the file, bandwidth or data processing constraints
Saving	Usually on the hard drive, also possible in *binary large object (blob)* fields of a database
Examples	Proprietary formatted files, archives, drive-images, execuTables, scripts, ...

Table 31: Type Details - Binary / Non-Multimedia File Types

Representation Examples

Name	Größe	Typ	Änderungsdatum
images	2 Objekte	Ordner	Do 28 Jul 2011 11:22:59 CEST
151619_Golf.jpg	26,2 KB	JPEG Image	Di 05 Apr 2011 07:46:22 CEST
OSBB_2011.png	535,0 KB	PNG-Bild	Do 07 Jul 2011 10:05:41 CEST
576207.pdf	14,2 KB	PDF-Dokument	Mo 24 Jan 2011 17:43:04 CET
config.xml	395 Bytes	XML-Dokument	Mi 23 Feb 2011 10:31:31 CET
H7.zip	246,6 KB	Zip-Archiv	Di 28 Jun 2011 10:45:20 CEST
MyFavourites.odt	20,9 KB	ODT-Dokument	Di 03 Mai 2011 17:40:07 CEST
PowerPointViewer.exe	25,8 MB	DOS/Windows-Programm	Mo 31 Jan 2011 11:40:36 CET
something.xlsx	9,2 KB	Excel-2007-Tabelle	Mi 06 Apr 2011 11:31:06 CEST
SuperDB.accdb	436,0 KB	Unbekannt	Mi 06 Apr 2011 17:24:46 CEST

Figure 44: Binary / Non-Multimedia - Representation Example 1

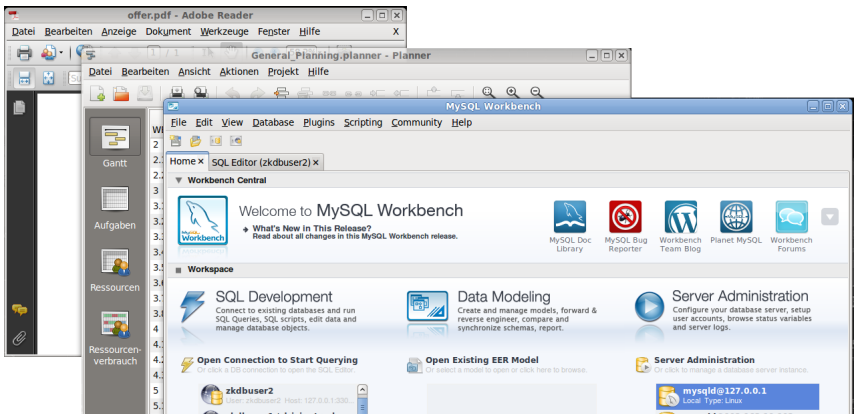


Figure 45: Binary / Non-Multimedia - Representation Example 2 – Proprietary File Formats

Reference

A reference is a pointer to an instance of a type or null.

Possible Values	pointer to another value, pointer to an address null, invalid
Formats	the format depends on the context of the usage
Order	none
Constraints	none
Saving	depends on the usage-context, often the ID of the references object/type is used
Examples	Link, Pointer to an address on the harddive or RAM, object reference in a programming language

Table 32: Type Details - Reference

Graphical Representations

#	productid	categoryid	name
1	feline01	CATS	Hairy Cat
2	feline02	CATS	Groomed Cat
3	canine01	DOGS	Medium Dogs
4	canine02	DOGS	Small Dogs
5	avian01	BIRDS	Parrot
6	avian02	BIRDS	Exotic
7	fish01	FISH	Small Fish
8	fish02	FISH	Large Fish
9	reptile01	REPTILES	Slithering Reptiles
10	reptile02	REPTILES	Crawling Reptiles

Figure 47: Reference - Representation Example 1

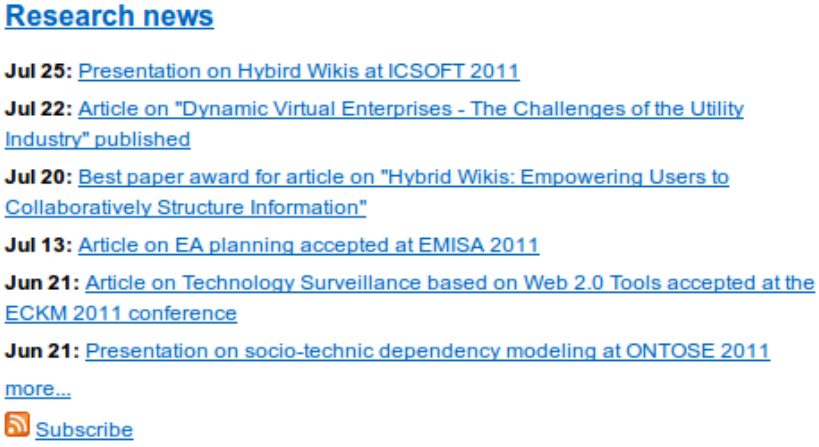


Figure 46: Reference - Representation Example 2[36]

36 This graphic is an extraction of a screenshot from the following page: http://wwwmatthes.in.tum.de/wikis/sebis/home, Accessed: July 28th, 2011 10:30pm.

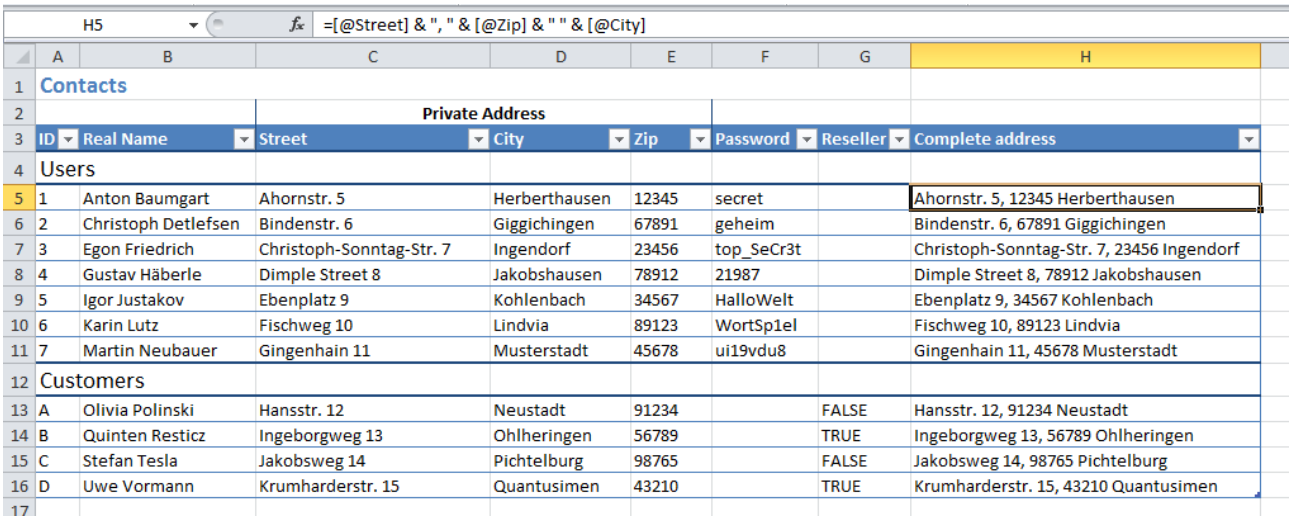
Function / Formula

A function or formula as it is called in Microsoft's Excel represents a calculation result. There are various ways of how a formular can be separated from other content, e.g. by using a special first character such as “=” or embedding the formula in a specified character sequence. Furthermore special placeholders for a function/formula can be inserted. They are often called *Fields*.

Possible Values	Boolean, Number, Text
Formats	varies context and content specificly
Order	none
Constraints	none
Saving	formula itself as text, formula as a template/macro for reuse, only the result of the formula
Examples	concatenating two strings, adding numbers, getting information from somewhere else

Table 33: Type Details - Function / Formula

Representation Examples



H5 =[@Street] & ", " & [@Zip] & " " & [@City]

	A	B	C	D	E	F	G	H
1	Contacts							
2			Private Address					
3	ID	Real Name	Street	City	Zip	Password	Reseller	Complete address
4	Users							
5	1	Anton Baumgart	Ahornstr. 5	Herberthausen	12345	secret		Ahornstr. 5, 12345 Herberthausen
6	2	Christoph Detlefsen	Bindenstr. 6	Giggichingen	67891	geheim		Bindenstr. 6, 67891 Giggichingen
7	3	Egon Friedrich	Christoph-Sonntag-Str. 7	Ingendorf	23456	top_SeCr3t		Christoph-Sonntag-Str. 7, 23456 Ingendorf
8	4	Gustav Häberle	Dimple Street 8	Jakobshausen	78912	21987		Dimple Street 8, 78912 Jakobshausen
9	5	Igor Justakov	Ebenplatz 9	Kohlenbach	34567	HalloWelt		Ebenplatz 9, 34567 Kohlenbach
10	6	Karin Lutz	Fischweg 10	Lindvia	89123	WortSp1el		Fischweg 10, 89123 Lindvia
11	7	Martin Neubauer	Gingenhain 11	Musterstadt	45678	ui19vdu8		Gingenhain 11, 45678 Musterstadt
12	Customers							
13	A	Olivia Polinski	Hansstr. 12	Neustadt	91234		FALSE	Hansstr. 12, 91234 Neustadt
14	B	Quinten Resticz	Ingeborgweg 13	Ohlheringen	56789		TRUE	Ingeborgweg 13, 56789 Ohlheringen
15	C	Stefan Tesla	Jakobsweg 14	Pichtelburg	98765		FALSE	Jakobsweg 14, 98765 Pichtelburg
16	D	Uwe Vormann	Krumharderstr. 15	Quantusimen	43210		TRUE	Krumharderstr. 15, 43210 Quantusimen
17								

Figure 48: Function / Formula - Representation Example 1

```
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod
tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At
vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren,
no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit
amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut
labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam
et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata
sanctus est Lorem ipsum dolor sit amet.

The current market share is <?php getMarketShare(); ?>.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod
tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.
```

Figure 49: Function / Formula - Representation Example 2

Other Types

The following types are only used in theory or in a specific context:

Type Name	Formats	Possible Values	Order	Examples
AutoValue (MS Access Name)	none	any	depends on the auto value	Increases the value each time the type is inserted
Entity type	any	any combination of primitive types	no order by default	Complex types
Identifier	any	any comparable data	no order	unique (primary) key
Nulltype	none	null	no order	null
Placeholder	none	any type in the scope	no order	Object

Table 34: Other Types - Details

List of Figures

List of Tables

Bibliography

Astels, David; Jeffries, Ron (2003): test-driven development: A Practical Guide, Pearson Education Inc., Upper Saddle River, New Jersey, USA 2003.

Buckl, Sabine; Matthes, Florian; Neubert, Christian; Schweda, Christian M. (2010): A Lightweight Approach To Enterprise Architecture Modeling and Documentation, CaiSE Forum 2010.

Büchner, Thomas; Matthes, Florian (2006): Introspective Model-Driven Development, Proc. of Third European Workshop on Software Architectures (EWSA 2006), pages 33-49, LNCS 4344, Springer, Nantes, France, 2006.

Ebersbach, A.; Glaser, M.; Heigl, R. (2006): Wiki Web Collaboration, Springer Verlag, Heidelberg, Germany 2006.

Kemper, Alfons; Eickler, André (2009): Datenbanksysteme: Eine Einführung, Oldenbourg Wissenschaftsverlag, München 2009.

Hansen, Hans Robert; Neumann, Gustaf (2005): Wirtschaftsinformatik 1: Grundlagen und Anwendungen 9. Auflage, Lucius & Lucius Verlagsgesellschaft mbH, Stuttgart 2005.

International Software Testing Qualifications Board (ISTQB) (2011): Certified Tester Foundation Level Syllabus, March 31th, 2011.

Kolberg, Michael (2007): Access 2007: Intelligentes Datenmanagement von Anfang an, Markt + Technik Verlag, München 2007.

Marcus, Aaron; Smilonich, Nick; Thompson, Lynne (1995): The Cross-GUI Handbook For Multiplatform User Interface Design, Addison-Wesley Publishing Company, Inc., USA 1995.

Matthes, Florian; Neubert, Christian; Steinhoff, Alexander (2011): Hybid Wikis: Empowering Users To Collaboratively Structure Information, ICSOFT 2011.

Münz, Stefan (2006): Professionelle Websites: Programmierung, Design und Administration von Websites 2nd edition, Addison-Wesley Verlag, Munich, Germany 2006.

OpenOffice.org (2009): OpenOffice.org 3.1 Developer's Guide, Public Document License 1.0, April 2009.

Scheller, Angela (1993): Informationsmodellierung für verteilte Anwendungen auf Basis standardisierter Datenmodelle, R. Poldenbourg Verlag, Munich, Germany/Vienna, Austria 1993.

Tapscott, Don; Williams, Anthony D. (2006): Wikinomics How Mass Collaboration Changes Everything, USA Portfolio, a member of Penguin Group Inc., USA 2006.

Ullenboom, Christian (2009): Java ist auch eine Insel: Programmieren mit der Java Platform, Standard Edition 6 8th revised edition, Galileo Press, Bonn, Germany 2009.

Walkenbach, John (2010): Excel 2010 Bible, Wiley Publishing Inc., Indianapolis, USA 2010.