D2.4

Reason Maintenance - Conceptual Framework

Project title:	**Knowledge in a Wiki**
Project acronym:	KIWI
Project number:	ICT-2007.4.2-211932
Project instrument:	EU FP7 Small or Medium-scale Focused Research Projects (STREP)
Project thematic priority:	Information and Communication Technologies (ICT)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	ICT211932/LMU-Munich/D2.4/D/PU/a1
Responsible editors:	Jakub Kotowski
Authors:	François Bry and Jakub Kotowski
Reviewers:	Josef Holý and Michael Zach
Contributing participants:	LMU-Munich
Contributing workpackages:	WP2
Contractual delivery:	1 June 2009
Actual delivery:	19 June 2009

Abstract
This paper describes the conceptual framework for reason maintenance developed as part of WP2.

Keyword List
reason maintenance, reasoning, explanation, rules

Project funded by the European Commission within the Seventh Framework Programme.

Reason Maintenance - Conceptual Framework

François Bry and Jakub Kotowski

[1] Institute for Informatics, University of Munich
Email: bry@lmu.de

[2] Institute for Informatics, University of Munich
Email: Jakub.Kotowski@ifi.lmu.de

18 June 2009

Abstract
This paper describes the conceptual framework for reason maintenance developed as part of WP2.

Keyword List
reason maintenance, reasoning, explanation, rules

Contents

1 A social perspective on knowledge representation and reasoning

Social software, such as Wikis, is affluent with user interaction and one of its main design goals is to support and facilitate the interaction and collaboration. We suggest that any advanced functionality, including reasoning, built into such kind of software should further boost this defining trait and we propose a contribution in the area. This section summarizes the distinguishing properties of social software and motivates the main features of reasoning, reason maintenance, and explanation planned for KiWi.

1.1 Work in progress

The work of knowledge workers is varied, doesn't follow strict guidelines and generally a great deal of it is a creative process, or work in progress, which doesn't adhere to predefined schemes (ontologies, scenarios) and may involve inadvertent inconsistencies. In the course of normal work, inconsistencies are unavoidable and will arise. In creative process they are even desirable. Creative thinking, by companies the often so sought-after "thinking outside the box", requires exploring alternatives, generating and testing ideas, or just brainstorming to get started. It is in the nature of the process that inconsistencies arise and can even constitute a desirable contrast that can be a source of inspiration which can help to proceed with the work. It is therefore desirable that the application supports the process and doesn't hinder it by enforcing unnecessary constraints. We suggest that in the case of our social semantic platform it means that ontologies should emerge and inconsistencies should be tolerated and users should be supported in discovering them and resolving them. Note, that it is the responsibility of users to eliminate inconsistencies; it is the responsibility of the software to draw attention to them and provide additional information about them (about their origin, users and facts involved, etc.) for example by means of explanation. It is also responsibility of the software to provide tools that can be used to "emerge semantics". We suggest that in KiWi these tools are informal, semi-formal, formal annotations, user-defined rules and reasoning.

1.2 Emergent collaboration - Joint work

Social software revolves around user collaboration. In an enterprise environment, rather than in the Internet, this collaboration may be more directed towards a common goal. Still, a free, unconstrained collaboration where virtual teams form spontaneously, based on need, is one of the main distinguishing features of social software. As McAfee remarks, it is, however, naïve to just provide users with a new software and expect that everyone will start using it and content will just emerge [22]. In other words, incentives are still needed, in an enterprise environment even more than on the Internet. We suggest that any semantic social software platform should facilitate joint work as much as possible and provide further incentives for active collaboration.

Users collaborate to achieve a common goal, they work together, often rather spontaneously, sharing and comparing their viewpoints which can frequently be in conflict. Conflicts can be seen as mutually inconsistent views that need to be settled. This, however, doesn't mean that they are not desirable. On the contrary. The same way as inconsistencies can drive a creative process, disagreements can be a means of creative problem solving [17]. It has been shown that collaborative reasoning (problem solving) can be more efficient than individual reasoning [15]. On the one hand, the study presupposes a live environment where people gather together in

a room and talk together, on the other hand the communication can, at least in part, happen vicariously in a social platform. Also, another study [20], about computer support of engineering design, indicates that with respect to means of communication less can sometimes be more:

> The most unexpected result of these experiments was that the participants who were limited to communicating with text-only explored three times as many ideas as those using graphic tools. Examination of the protocols suggested that the participants using the graphic tools fixated on solutions and gave themselves to exploring the early solutions and often engaging in lower level, even trivial, design decisions while ignoring the opportunity to explore more widely. Consequentially, the students using graphic communication explored the problem space less thoroughly than those using text.

We suggest that a semantic social software platform should support users in expressing their agreements and disagreements explicitly as well as to provide them with tools facilitating discovery and explanation of disagreements.

1.3 Knowledge representation and reasoning the Wiki way

Knowledge representation and reasoning in KiWi should be easy to use and grasp in order to retain the simplicity of a Wiki. Traditionally, knowledge is represented in complex formalisms by experts cooperating with domain experts. Such an approach is neither possible nor desirable in an enterprise semantic social platform. It should be possible even for a motivated casual user to semantically enrich content. The approach we suggest to take is to provide users with a means of creating annotations which can be made gradually more formal and precise as users become familiar with the system. One challenge is to knit the annotations together with reasoning and to maintain simplicity of the whole. To meet this challenge, we suggest building on simple things: "structured tags", a form of "semi-formal annotations", to bridge the gap between informal annotations (tags) and formal annotations (RDF) and a rule-based inconsistency tolerant reasoning complemented by reason maintenance to provide simple reasoning which can easily be explained.

1.4 Emerging semantics: from informal to formal knowledge representation

Social platforms teeming with user activity enable not only emergent collaboration but also emergent semantics. Traditional approaches to knowledge management impose a rigid prior structure on information. In contrast, collaborative tagging leads to emergent creation of folksonomies[1] advantages of which over predefined taxonomies have been discussed before. We argue that, given proper tools, user collaboration can lead also to semantically richer structures. Folksonomies can be made explicit using specialized algorithms based on data-mining and information extraction methods. This kind of processing is necessary because users can use only a very limited tool - simple tags. Structured tags and rules provide users with a much more powerful tool which, in its simplest form, can be used the same way simple tags are used and, in its full power, enables direct creation of ontologies precisely specified in RDF/S. Structured tags provide more control over relations between tags and rules allow for a translation to

[1] We use the word *folksonomy* to mean a user-generated taxonomy. Folksonomy is implicitly present in the bag of all tags but can also be extracted and represented the way traditional taxonomies are represented.

RDF/S once the meaning of tags emerges (a user community agrees on it), see Section 4 for details.

1.5 Negative information

In real life, people often say and need to say that something is not something else, something is not possible, it is excluded, or not allowed; they are used to working with negative information. We suggest negative tagging as a way to express negative information in KiWi. To see how this could work, suppose there is a proposal for a project that should be reviewed and evaluated. Someone can tag it as "*–risky*"[2], to express that in his or her opinion that the project is not risky. A manager could be interested in finding all projects that are not risky and have not been reviewed yet. In this query there are two different kinds of negative statements. "*Projects that are not risky*" refers to explicitly assigned negative tags "*–risky*", whereas "*Projects that have not been reviewed yet*" refers to a missing tag "*reviewed*". We argue that knowledge workers will benefit from having both these negations as they increase expressivity of the knowledge representation formalism.

1.6 Inconsistency tolerant reasoning

A manager trying to determine potential problems with projects may be interested in finding disagreements about riskiness of a project. In other words, it is interesting to know whether there are some projects tagged both as "*risky*" and "*–risky*". The system should know that these two tags exclude each other and should be able to inform the user about it. At the same time, reasoning has to work even in presence of such inconsistencies. This leads to a need of a kind of inconsistency tolerant reasoning which is described in more detail in later sections.

2 What the user interacts with

It is necessary to know what the user-level concepts of a system are in order to design the system properly. This section reviews the basic concepts a user will interact with within the KiWi system. It is based on work published in [5] and most of it is deliberately shared with the deliverable [9] as it provides a common ground for querying and reasoning and other parts of the KiWi system.

2.1 Content

This section outlines the representation of content in the KiWi Wiki. "Content" here refers to text and multimedia which is used for sharing information, most frequently through the use of natural language, between the users of the Wiki, and whose meaning is not directly accessible for automatic processing. Information Extraction techniques enables the computerised processing of structured data extracted from text or speech, but this introduces another level of representation which is not considered "content" in this sense.

[2]We suggest to use the minus sign for assigning a negative tag, the word "not" is reserved (in line with tradition) for negation as failure. See Section 3 for details about negations.

2.1.1 Content Items

Content items, the primary unit of information in the KiWi wiki, are composable, non-overlapping documents. Every content item has a unique URI and can be addressed and accessed individually. As a consequence, there is no inherent distinction between Wiki pages and content items. Rather, by default, all content items are Wiki pages.

An atomic textual content item can be thought of as being similar to a paragraph or section in a formatted text in that it contains content but also structures it. A content item can directly contain only one type of content, for example text or video. However, content items can be nested through transclusion [23] to enable the representation of complex composite content structure. Consequently, a content item may contain its (textual or multimedia) content and any number of inclusion declarations, both of which are optional.

Having an explicit concept of content structure in a Wiki is desirable both with respect to the semantic as well as the social nature of a Semantic Wiki; the structural semantics of the content can be immediately used for querying and reasoning, for example for automatically generating tables of contents through queries, as well as for facilitating collaboration and planning of content. In addition, content items constitute a natural unit for assigning annotations for content (see Section 2.2).

Content items can be multiply embedded in other content items. This means that content items can be easily reused and shared, which is useful for example for schedules or contact data. If only a copy of the content item's content is embedded, multiple occurrences of the content item in the Wiki cannot be traced as naturally or easily and data, for example changes of schedule or email address, have to be updated in multiple locations. On the other hand, updating a multiply embedded content item or reverting it to an earlier version can lead to unintuitive and undesired effects when the content item changes in all contexts it is embedded in without the editing user being aware of all these contexts. Therefore, upon modifying a content item that appears in several different locations, the user is presented with a list of the embedding locations and the choice to edit the content item or a copy its content.

Loops arise when a content item contains itself as a descendant through multiple embedding. The resulting infinite recursion is problematic with respect to the rendering of the content item[3] as well as reasoning and querying over it. Since loops additionally arguably have no straightforward meaningful interpretation in the Wiki context, transclusions which would cause loops are generally forbidden. However, it is generally possible to embed a content item several times into another content item or one of its descendant content items as long as the embedding does not give rise to a loop.

Assuming that all links to content items included through transclusion have been resolved, the Wiki content can be seen to consist of a set of finite trees. Root nodes, that is, content items that are not contained in another content item, then have a special status in that they encompass all content that forms a cohesive unit. In this, they can be seen as being alike to a Wiki page in a conventional Wiki.

2.1.2 Fragments

Fragments are small continuous portions of text (or, potentially, multimedia) that can be annotated. While content items allow the authors to create and organise their documents in a modular and structured way, the idea behind fragments is that they enable the annotation of

[3] At least if we assume that all of the content item is to be rendered at once.

user-defined pieces of content independently of the canonical structure. If content items are like chapters and sections in a book, then fragments can be seen as passages that readers mark; they are linear and in that transcend the structure of the document, spanning across paragraphs or sections. Different sections of the book might be marked depending on which aspect or topics a reader is interested in.

Fragments should be maximally flexible in their placement, size and behaviour to allow for different groupings. Towards this goal, it is generally desirable that – unlike content items – fragments can overlap. The intersection between two overlapping fragments can either be processed further or can be ignored. When two overlapping fragments f_1 and f_2 are tagged with "a" and "b" respectively, a third fragment that spans over the overlapped region and is tagged "$a,\ b$" could be derived automatically. Similarly, automatically taking the union of identically tagged overlapping or bordering fragments might be intuitive and expected by the user. However, this automatic treatment of fragments is a complex issue which might not always be appropriate or wanted.

Therefore, fragments are seen as co-existing but not interacting, meaning that the relationships between fragments are not automatically computed and no tags are added. This view has the advantage of being simpler and more flexible in that control of the fragments and their tags stays with the user. It is also in tune with the philosophy that, unlike content items that always only realise one structuring, fragments are individual in that different users can group a text in many different ways and under many different aspects.

Fragments can either be restricted to be directly contained in one content item, or they can span across content items. In the latter case, a rearrangement of content items can lead to fragments that span over multiple content items which no longer occur in successive order in the Wiki and, similarly, transclusion means that content items may contain only part of a fragment with the other part being absent (but present in some other content in which the content item is used).

To avoid these problems, in the KiWi Wiki, fragments start and end in the same content item, but can span over contained content items. One single content item then contains the whole fragment. More precisely, this means that fragments can span over different content items as long as they have a common parent content item and all interjacent sibling content items are part of the fragment, too.

Two possibilities of realising fragments are the insertion of markers in the text to label the beginning and end of an fragments ("intrusive"), or external referencing of certain parts of a content items, using for example XQuery, XPath, XPointer, or line numbers ("non-intrusive"). As the former means that fragments are less volatile and updates to the text do not affect fragments as easily, for example when text is added to the fragment, fragments in the KiWi Wiki are intrusive.

2.1.3 Links

Links, that is simple hypertext links as in HTML, can be used for relating content items to each other and to external resources. Links have a single origin, which is a content item, an anchor in this origin, and a single target, which is a content item or external URI. Links can be annotated.

2.2 Annotation

Annotations are meta-data that can be attached to content items, fragments and links and which convey information about their meaning or properties. Annotations can be assigned by the user, either manually or via rules. However, content items and tag assignments also have system meta-data such as the creation and last edit date and the author(s) of a content item or tagging. These meta-data are realized in the form of automatically generated annotations which cannot be modified by the user. The Wiki comes with a pre-defined, application independent RDF/S vocabulary expressing authorship, versions, and the like. This is not further developed in the following as it is not directly relevant to the KWQL query language or reasoning. The text in the following focuses on user-generated annotation.

Two kinds of annotations are available: tags and RDF triples. Tags allow to express knowledge informally, that is, without having to use a pre-defined vocabulary, while RDF triples are used for formal knowledge representation, possibly using an ontology or some other application-dependent pre-defined vocabulary. Regular users are generally not confronted with RDF, which is considered an enhancement for experienced users, but the KiWi Wiki aims to allow for a smooth transition between informal and formal annotation as will be described in the following.

For practical applications, support for RDF is important in order to enable of interoperability with current Semantic Web and linked data [2, 1] applications.

2.2.1 RDF/S

"The Resource Description Framework (RDF) is a language for representing information about 'resources' in the World Wide Web." [21]. It is currently the most common format for semantic data. RDF data are suited for processing by machines but not easily human-interpretable. For practical applications, support of RDF is important in order to enable of interoperability with current Semantic Web and linked data [2, 1] applications.

"RDF's vocabulary description language, RDF Schema, is a semantic extension [...] of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources." [3]. These basic constructs can be used to describe ontologies.

An ontology is usually defined as an explicit specification of conceptualization [16]. "A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose." [16]. The representation can be done for example using the RDF/S or OWL language. The KiWi system chooses the RDF/S language to specify its ontologies because it is in many ways simpler than OWL and yet it is sufficient for most purposes in KiWi.

The KiWi Wiki will translate flat, structured tags, and links to an RDF/S representation using a predefined vocabulary (ontology) in order to make them accessible to semantic querying and reasoning.

2.2.2 Tags

One problem that frequently arises in the context of Semantic Web applications that use relatively complex formalisms such as RDF for annotation is that it is hard to motivate users to annotate content since they find the process complicated, laborious and time-consuming.

In addition, ontologies are not ideally suited for capturing emerging knowledge in a social application as they constitute a top-down, single-viewpoint approach to describing domain concepts and their relations, and do not necessarily account for newly arising concepts whose

definition and classification might initially be vague. However, Semantic Wikis are applications for collaborative, work in progress. As such, not only content but also annotations and their semantics should be easy to use and flexible, account for multiple co-existing and interacting viewpoints and gradually refinement and should arise in a bottom-up manner.

So, while RDF backed up by ontologies is a powerful and established Semantic Web tool, Social Semantic Software also needs less formal and more flexible modes of annotation such as tagging, the assignment freely of chosen keywords. The ability to tag content is provided by many social websites where users can assign tags to content, for example bookmarks or pictures, thus creating groups of items that share similar characteristics, namely a relation to the tag which serves as the label for that group. Through this, ordering and retrieval of content is enabled, serving as an incentive for tagging. Additionally, in many applications, tags are pooled across users, meaning that a user can see both which tags have been assigned to a resource by other users as well as which items a given tag has been used on. The aggregated collection of tags used in a system is referred to as a folksonomy, a portmanteau word made up of folk and taxonomy. Unlike a taxonomy, a folksonomy, however, does not natively have a hierarchical structure (or, for that matter, any structure at all).

In a tagging system, no meaning is assigned a priori to a string, and on the other hand, no specific string is pre-assigned to express a concept. The meaning of a tag can thus be seen to emerge from the collaborative tagging by individual users who do not have to commit to a pre-defined view on categorization.

However, as a consequence of this flexibility and accommodation for different points of view, the same concepts are frequently expressed using different tags, for example synonyms, tags in different languages or inflected or simply differently spelt versions of the same word. Similarly, different concepts may be assigned the same string as a tag due to polysemy and homographs. Yet another problem is basic level variation, varying degrees of specificity in tagging depending on the level of expertise in an area. Since no explicit relations between tags exist in a folksonomy, tags referring to similar or identical concepts at different degrees of specificity are not recognized as being related.

These factors have a negative influence on the consistency of the emergent semantics and consequently on both recall and precision as well as users' understanding of the structure of the system. The practical and social aspects are interrelated and mutually reinforcing since users not sure about tagging conventions are likely to break them, leading to more inconsistency in the folksonomy.

As above observations indicate, social and technical factors must be taken into consideration – but can also be leveraged – when supporting the process of emerging semantics in a folksonomy.

The KiWi Wiki employs tagging with advanced features to help overcome the downsides of uncontrolled, ad-hoc categorization and to enable a transition between and a combination of informal and formal annotation, that is, RDF.

Tags as concepts. The KiWi Wiki distinguishes between tags as mere strings or labels and the abstract concepts they stand for by representing the tag concepts as content items, tags in our terminology. Each tag is associated with one or more labels which may overlap between tags, thus mirroring the non-unique mapping between strings and concepts as described above. Each tag concept content item may, but does not have to, contain a textual description of its meaning and a list of resources it has been assigned to, thus making it easier for users to negotiate the meaning and use of the concept.

When a user enters a tag to be assigned to a resource, the system automatically resolves

it to the corresponding tag concept, allowing the user to intervene if she disagrees with the concept association that occurs in the case of ambiguity. Various approaches for semantic disambiguation, for example based on co-occurences of concepts and distances between words in Wordnet, exist and could be used. Each tag concept's content item can be used as a tag label, meaning that each concept can be unambiguously addressed during tagging, even when all of its associated labels are ambiguous. This is especially relevant for the automatic assignment of tags where no user-intervention is possible or desired. In summary, a resource in the KiWi Wiki is tagged not with a string or label but with a concept to which the label serves as a shortcut.

Besides offering a natural solution for resolving ambiguous mappings between concepts and labels, tags can also be matched to formal concepts in an ontology, allowing for the ontological grounding of a tag concept.

Note that, when no resolving of ambiguous tag labels is available or desired, the system described can adjusted for this by requiring tag labels to be unique.

Tagging The assignment of a simple tag, a tagging, in the KiWi Wiki means the association of a content item, fragment or link with a tag using a tag label. The assignment itself additionally is connected to maintenance information.

A tagging is thus a tuple consisting of a tag, a tag label, a tagged resource, and maintenance information needed for processing of taggings. Maintenance information includes information about the origin of the tagging (e.g. whether it was assigned by a user or based on which information it was derived), the date when the tagging was created, a marker which allows to distinguish between explicit, implicit, and system taggings, and the tag label using which the tagging was created. Tagging can be either positive or negative. This is distinguished by a *polarity property*. Negative tags are displayed with a minus ("–") sign directly in front of their tag label.

Tag assignments can be explicit, that is, done manually by a user, or implicit when they are inferred by a reasoner based on user-defined rules.

Tag label normalization Tag normalization defines an equivalency on the set of tag labels. Tag label normalization can mean for example that the tag labels "Wifi", "WiFi", "Wi-fi" and "WIFI" are all equivalent to the tag label "wifi" which can be the canonical form of this tag label equivalency class. The canonical form is shown in aggregated views such as tag clouds. Trailing whitespace and multiple whitespace within a tag are always removed upon saving the tag. In other cases, each user always sees the tag label that he or she entered. Further, tag label normalization in KiWi is done by converting all letters to lowercase and removing punctuation.

Negative tags. In a collaborative context, we may be interested in tracking disagreements which presupposes some way to express negative information. Just as a user can tag a resource with tag "t" he or she may want to tag it with "–t" as a way to express disagreement or to simply state that the resource is not "t" or does not have the property "t". An example may be a medical doctor tagging a patient's card as "–lupus" to state that the patient definitely does not have "lupus".

Although a tag "–t" could be seen as introducing classical negation into the system, it may in fact be only a very weak form of negation because we can allow negating only pure tags, not general formulae (or sets of tags), and the only way to interpret this kind of negation would be by introducing a rule which says that from tag "t" and tag "–t" a contradiction symbol should be derived. For more about negative tags see Section 3.

Structured tags. Structured tags can be used to enhance the expressive power of tags and as an intermediate step between informal (atomic tags) and formal (RDF triples) annotation.

Two basic operations lie at the core of structured tagging: grouping and characterization. Grouping, denoted "()", allows to relate several (complex or atomic) tags using the grouping operator. The group can then be used for annotation. Example: a Wiki page describes a meeting that took place in Warwick, UK on May 26, 2008, began at 8 am and involved a New York customer. Using atomic tags, this page can be tagged as "Warwick", "New York", "UK","May 26", "2008", "8am" leaving an observer in doubts whether "Warwick" refers to the city in UK or to a town near New York. Grouping can be used in this case to make the tagging more precise: "(Warwick, UK), New York, (May 26, 2008, 8am)". The annotation of a group of tags assigns only the higher unit made out of the individual atomic tags or group, but not the separate constituents. If this functionality is desired, it could be implemented for example via rules. A group of tags can also be used to describe the properties of something whose name is not yet known or for which no name exists yet.

Characterization enables the classification or, in a sense, naming of a tag. The characterization operator, denoted ":", can be used to make the tagging even more precise. For example, if we wanted to tag the meeting Wiki page with a geo-location of the city Warwick, we could tag it as "(52.272135, -1.595764)" using the grouping operator. This, however, would not be sufficient as the group is unordered. Therefore we could use the characterization operator to specify which number refers to latitude and which to longitude: "(lat:52.272135, lon:-1.595764)" and later perhaps specify that the whole group refers to a geo-location: "geo:(lat:52.272135, lon:-1.595764)". Similarly, Warwick in our example could be expressed as "location:(warwick)" to differentiate it from Warwick fabric or person with the last name Warwick.

Together, grouping and characterization provide a powerful tool for structuring and clarifying the meaning of tags. Structured tags are a step between informal simple tags and formal RDF/S annotations.

The meaning of a structured tagging rests, for the most part, on the user who specified it. Structured tags do not impose strict rules on their use or purpose, they only allow users to introduce structure into taggings. It can be seen as a Wiki-like approach to annotation which enables a gradual, bottom-up refinement process during which meaning emerges as the user's work and understanding develop.

Minimal rules are, however, necessary in order to ensure a flexible, useful order and to avoid destructive chaos:

- Groups
  - can be used on positive and negative atomic tags and groups,
  - are unordered,
  - cannot contain two equal members, e.g. (Bob, Bob, Anna) and ((Bob, Anna), (Anna, Bob)) are not allowed,
  - can contain arbitrarily but finitely many elements,
  - can be arbitrarily but finitely nested,
  - are identical in meaning to the atomic tag when they only contain one element, i.e. (Anna) is the same as Anna
- Characterization

  - can be used on positive and negative atomic tags and groups,
  - is not commutative, i.e. geo:x is not the same as x:geo.
  - can use atomic and complex tags as a label

Structured tags have to be syntactically correct. That means that for example "Bob:190cm,90kg" is not a valid structured tag.

The same information can be expressed in many ways using structured tags. The above described way of structuring geo-location is only one of several. Others include:

- geo:(x:y):(1,23:2,34)
- geo:(y:x):(2,34:1,23)
- geo:(1,23:2,34)
- geo:1,23:2,34

and possibly many more. Users are free to choose what suits their needs and purpose the best. Of course, for structured tags to be useful in a community, the community needs to agree on a common way of structuring information. It is important that only a very minimal structure is required by the system leaving the possibilities open; different users and different communities can agree on different ways of structuring the same information. This heterogeneity is an advantage. First, it provides users with freedom and second, it can be beneficial for different communities to encode similar kinds of information in different ways as the precise meaning of a similar concept may differ. In such a case, different encoding may facilitate automatic translation of structured tags to formal annotations because it would allow to distinguish the two concept structurally.

Mapping tags to RDF. Simple and structured tags can be represented using RDF in KiWi. The representation of simple tags is a straightforward translation of a tagging n-tuple to a set of RDF triples:

- Tagging
  - hasAuthor Author
  - hasTag Tag
  - hasItem Taggable
  - hasPolarity one of 'positive', 'negative'

Where Author is the author of the tagging (either a user or the KiWi system), Tag is a class representing tags (that is, it is a ContentItem possibly with additional properties), Taggable is a class of items that can be tagged (includes e.g. ContentItem and Link).

Structured tags are represented in RDF the following way:

- StructuredTagging
  - hasAuthor Author
  - hasItem Taggable
  - hasStructuredTag one of Grp, Char

* Grp hasMember one of Grp, Char, Tag
* Char hasLeft one of Grp, Char, Tag
* Char hasRight one of Grp, Char, Tag
* each Char has exactly one hasLeft and exactly one hasRight property

It is a direct syntactic translation. Note that only positive simple tags are allowed inside structured tags. Negative tags may be useful inside structured tags as well: "(red, nice, –sweet)" but their meaning is not clear. Consider the following examples: "sweet, (red, nice, –sweet)", "(sweet, –sweet)", "apple:(ripe:sweet, crude:–sweet)", "–crude:sweet", etc. Although the intuitive meaning of simple negative tags is clear and well defined with respect to simple tags, it is not yet clear what the meaning of negation should be with respect to structured tags.

Tags and content structure. As KiWi will support composed content items and fragments, it is a question whether annotations should be propagated from items and fragments to containing content items or the other way around. Propagation may be appropriate in cases where the parent-child relationship corresponds to a "isA" or "isInstanceOf" between concepts which the content items describe. In contrast, when the parent-child relationship corresponds to an "isPartOf" relation then tag propagation may be not desirable. The content structure will be accessible to the rule and query language in form of metadata. Therefore, it will be possible to define rules which specify the type of desired tag propagation. There can even be different rules for different parts of the Wiki thanks to the support of content-item rule scope (see Section 5.1.2).

The social weight of tags. If several users tag one item with the same tag, it is useful to aggregate these tag assignments to give a clearer view of all the tags assigned and derive information about the popularity of the tag assignment. Tag assignments then can be seen to have weights. On the other hand, other users might not agree with the assignment of a certain tag to a content item, and add a negative component to the tags' weight to express this. The overall social weight of a tag can then be calculated by assigning a value to both a tag assignment and disagreement with it and calculating the total. The social weight of a tag then summarizes the users' views on the appropriateness of a specific tag assignment and thus provides a valuable measure that can be used in reasoning and querying. Note that agreeing with a tag assignment is identical to also making it but disagreement is not identical to adding the negative tag since not being content with the assignment of tag does not necessarily imply that one thinks its negation makes an appropriate tag.

Further, the tag weight gives an overview over users' opinions and could help form a consensus on tag usage. TODO(CITE) show evidence that users have a desire for consistency with other users' tag assignments and are often willing to adopt a tag that they know has been used on the same resource by other users. Weighting tags could further foster this process. To facilitate this, it should be as easy as possible to also assign a tag or to disagree with its assignment.

Finally, reinforcing or disagreeing about tag assignments constitutes a low-barrier activity in the Wiki, making it easy for beginning users to participate.

Tag weights are also leveraged for enabling agreement and disagreement votes for content items. For every content item, there are two reserved tags, '%agree' and '%disagree' which are rendered as buttons or links below the content item. They are used to indicate agreement or disagreement not with tags but with the content of the content item.

The semiotic triangle. One question to ask when designing a system that includes annotations is "What is it that is being annotated?" On the purely technical level, this question may have a quick, superficial answer: "Any resource that the system allows to be annotated." But what does that mean precisely? Let us say that the resource is a content item about an elephant. Does a tag added to the page state a fact about the page itself (a representation of an elephant in the Wiki system) or does it refer to the actual elephant? This leads to a concept known as the semiotic triangle [24], Peirce's triad[26] or de Saussure's distinction between the signifier and the signified [28]. This distinction is important because it may have consequences on how annotations are interpreted and treated. In [25], the authors let the users explicitly express whether an annotation refers to a text or its meaning by providing them with a syntax that allows the users to distinguish between these two cases.

In tagging as implemented in the KiWi Wiki, this problem is, in fact, doubled: the tag and the tagged content item (the signifiers) each have an associated signified (or several signifieds depending on who is viewing the page, in what context the page appears, etc.). Through this mechanism, the KiWi system addresses the problem of multiple meanings of a tag label as described above. It, however, purposefully does not address the problem of what a tagging refers to – to a concept or the page that describes it. The reasons for this are that such a distinction is likely to be insignificant in most cases with respect to the practical use of tags, irrelevant to many users and, above all, forcing users to be aware of the distinction and employ it during tagging reduces the user-friendliness of tagging and thus deters users from annotating content. In summary, introducing such a distinction would be likely to lead to an unnecessary increase of complexity of the tagging process.

2.3 Social Content Management

Groups. User groups can be used among other things for personalisation of Wiki content, for querying and reasoning and to attribute Wiki data to a group. Tags are an easy way to group things which is used in the Wiki, so it is an obvious choice to form user groups by tagging users' content items. Groups are created by assigning at structured tag with the label 'group' to a content item, fragment or link, e.g. "group:(java)"

Access Rights. Users, user groups and rules for reasoning could be used to handle access rights in the Wiki, but this is a complex issue which requires further investigation. Questions that arise include who owns the rules and what are the access rights on rules and who can assign the tags that restrict the access. Static rules would not be suited for rights managements in all environments. For them to function well, the organization and roles in the Wiki have to be relatively stable, which may be the case in professional applications. In other areas, such as the development of open source software, such rules may not be desired or the social organization might not be static enough for rules to be adequate.

3 Negation

The two kinds of negations introduced in Section 1.5 amount to a kind of classical negation for explicit statements and to negation as failure to refer to missing information. Combination of classical negation and negation as failure would, however, significantly increase the complexity of the language and would not be easy to use for casual users. Therefore, we propose a very weak

form of classical negation in the form of negative tagging. It should be possible to integrate this kind of negation with negation as failure in a simple and user-friendly manner.

3.1 Negative tags

There can be different reasons for which a user may want to express negative information. One is to express exclusion, e.g. "This document is *not official*." Another one is to express disagreement, e.g. "Someone tagged a content item as *finished* and someone else disagrees and wants to say that the page is *not finished*." This section is concerned with the first type of negative information. The latter type is closely related to social weights of tags and is discussed in deliverable D2.2 [9].

Let us imagine that a user wants to say that the application behaviour described in a Wiki-page *is not a bug*. As already discussed, it is natural to expect that the system would bring the user's attention to inconsistent cases, such as when a page is tagged as "*–bug*" and "*bug*" at the same time. The system can detect such inconsistencies using a special constraint rule:

$$\$X \wedge -\$X \rightarrow \bot$$

Because this rule is the only way negative tags are interpreted[4] and because only tags, not arbitrary formulas, can be negated, a very weak form of negation is obtained. The "–" sign amounts to a negative marker interpreted by the constraint rule and nothing else. No *ex falso quodlibet* (the principle of explosion) holds because $\bot$ cannot stand in the head of a rule. Also, for example, the contrapositive of a rule does not apply. To illustrate this, imagine there is a page tagged "*–rock*" and there is a rule $artRock \rightarrow rock$. Then the contrapositive rule $-rock \rightarrow -artRock$, which holds in classical logic with "–" being classical negation, does not apply because, again, in our case the "–" sign is only a negative marker interpreted only by the above constraint rule.

The advantage of this approach is that it is possible to express negative information while keeping the problem of combining two kinds of negation easy to grasp. Note that the negative marker can stand only directly in front of a tag; i.e. it cannot stand in front of a formula. Hence, the rule language allows to refer to positive facts ("f"), negative facts ("$-f$"), missing positive facts ("$not\ f$"), and missing negative facts ("$not\ -f$"). This may need some explanation to users but we believe it is conceptually rather simple.

To explore the intuitive meaning of the negations consider the following scenario. Let us imagine there are two rules:

$$r_1 : budget \wedge not\ reviewed \rightarrow needsReview$$

and

$$r_2 : needsReview \rightarrow -reviewed$$

Then, if there is a page tagged as "*budget*" and it does not have the tag "*reviewed*" then it will be tagged as "*needsReview*" according to rule r_1. According to rule r_2 the page will also be tagged as "*–reviewed*". We now may ask what happens if a user eventually tags the page as "*reviewed*"? It would be natural to expect that the "*needsReview*" tag is removed and subsequently the tag "*–reviewed*" is removed as well leading to a consistent set of tags "*budget, reviewed*".

[4]This is true with respect to user-defined rules about tags. Greater care must be taken in combination with RDF/S annotations, see Section 3.2.

The behaviour could be different though. An implementation could first detect the contradiction following from tags "*reviewed*' and "*–reviewed*" and the above constraint rule. This could happen for two reasons: either the implementation is naïve or it could be a result of trying to make the system responsive at all times by postponing and queuing some operations. In any case, increased caution by implementation is necessary.

Agreements and disagreements. Before exploring combinations of negations, let us take a closer look at two problems that may arise when modelling agreements and disagreements using rules negative tags. First, imagine there is a rule

$$-agree \rightarrow disagree$$

If someone tags a page as "*–agree*" then it is according to this rule also tagged as "*disagree*". Then if someone tags the page as "*–disagree*" an inconsistency is detected. The problem is that in natural language "not disagreeing" is usually not perceived the same as "agreeing" but rather as an indeterminate opinion. The source of this possible confusion would be immediately visible to the user because the system would claim that "*–disagree*" and "*disagree*" are inconsistent and the user could see that the "*disagree*" tag is derived based on the user-defined rule. Therefore, in this case, the user could blame himself/herself for specifying a rule which does not capture his/her intuition correctly.

The second problem could be that an agreement and a disagreement with a specific thing by two different people does not necessarily need to be seen as an inconsistency in the sense of contradiction ("X is white", "X is not white") but rather as a mere difference in opinions ("UserV says X is white", "UserW says X is not white"). There are two answers to this problem. First, it is our modelling choice to capture agreements and disagreements the first way which makes inconsistency detection more applicable because in the other case only statements of a user could contradict other statements of this (and no other) user. Second, the approach could be extended to allow users (or system administrators) to specify when which rule applies with respect to the origin of facts, refer to Section 5.1.2 to see how.

3.2 Negative tags and RDF/S

The previous section discussed negative tags with respect to user-defined rules about tags. If such rules are combined with general RDF/S annotations and ontologies greater care must be taken to ensure that the meaning of negative tags is correctly combined with RDF/S semantics. Consider the following example:

$$page \quad hasTag \quad jaguar$$

$$page \quad hasTag \quad -animal$$

$$jaguar \quad rdfs:subClassOf \quad animal$$

In this case, the tagging of the page "*page*" and the last triple together express that on the one hand jaguars are animals and that the page is a jaguar but not an animal. Similar problem could arise with the rdf:type property. On the one hand, this combination of triples is not really inconsistent because it is only a human interpretation which takes a tag assignment as type or a class specification, on the other hand, users could intuitively expect this combination of triples to be marked as inconsistent.

3.3 Examples

The following examples show the possible combinations of negative tags and negation as failure.

$functionalRequirements \rightarrow requirements$
: "Functional requirements are requirements."

$-bug \rightarrow confusingFeature$
: "If someone feels the need to explicitly say that something is not a bug then perhaps it is a confusing feature."

$notfixed \rightarrow todo$
: "If something has not yet been tagged as fixed then it is to be done."

$bug \wedge not - regression \rightarrow checkRegression$
: "If something is a bug and it has not been verified yet if whether it is a regression (the "*–regression*" tag has not been assigned yet) then check for regression."

Note, that the combination "$-\,not\,regression$" is not allowed because the negative marker can stand only directly in front of a tag.

4 An approach to knowledge representation - emergent semantics

KiWi supports three kinds of metadata: informal (tags), semi-formal (structured tags), and formal (RDF/S annotations). Structured tags allow users to enrich ordinary tags by structuring them in order to clarify their context or qualify the thought statement that the tag is supposed to express. These structural hints can then be used to translate structured tags to formal RDF/S annotations via rules that specify their mapping to a predefined ontology. The advantage is that the simplicity of tagging is preserved: users can first tag and make the tagging more precise only later and eventually map it to a formal annotation. The approach could thus be summarized as "Tag, Think, Qualify, Map" – a cycle consisting of four steps. Let us take a look at an example.

Consider an enterprise Wiki in which each employee has a profile page. Alice, a junior assistant from the HR department of a company, has just hired a new employee, Bob. She creates a new page in the Wiki for Bob and decides to tag it with a few pieces of information about him. She assigns two tags to the page: "*manager*" and "*programmer*" because it is what came to her mind first. Later, when she comes back to the page, she realizes that the tagging can be confusing. Therefore she qualifies the tags to make it apparent that Bob is a manager with a programming background: "*position:manager*", "*experience:programmer*". Eva, a more experienced colleague of Alice, notices Alice's tags and lets her know about a new Wiki feature that their IT team is about to introduce. It is an org-chart widget that, given a name of an employee, displays the name of his or her manager and employees which report to him or her. The widget would be a nice addition to profile pages, so Alice and Eva talk to a contact from the IT department to ask how to include it. It turns out to be rather easy. The Wiki will automatically display the org-chart on any user page. User page is a page with type ucont:Employee defined in the company's UCONT[5] RDF/S ontology. Therefore all it takes to

[5] UCONT stands for "use-case ontology."

use the widget is to create a new rule in the Wiki: $position : manager \rightarrow rdf : type\ ucont : Employee$. It translates each structured tag "*position:manager*" to a formal RDF/S annotation "*rdf:type ucont:Employee*". Alice and Eva don't have to learn anything about ontologies and RDF/S and they still can indirectly enrich the Wiki content with formal annotations. Maybe, in the future, another widget will be able to summarize skills of an employee in which case Eva and Alice will benefit from their distinguishing between e.g. "*experience:programmer*" and "*position:programmer*".

This admittedly simplified example indicates how knowledge can emerge from initial tagging and user collaboration inside and outside a Wiki environment by taking advantage of structured tags and rules. Rules could of course be more sophisticated, e.g. so that users can tag only with the most specific tag and the more general ones are automatically derived. In the above scenario users could add the rule: "$position : manager \rightarrow employee$". At this point, Eva or Alice may realize that if they used structured tags of the form "type:NameOfTheType" then they would be able to define only one general rule: "$type : \$TYPENAME \rightarrow ucont : \$TYPENAME$" and use simple structured tags to specify RDF/S types from the company's predefined ontology. Note, that in a similar way, user taggings could be enriched with information from e.g. a SKOS thesaurus using rules such as: $(P\ hasTag\ T) \wedge (U\ skos : broaderThan\ T) \rightarrow (P\ hasTag\ U)$ and $(P\ hasTag\ T) \wedge (U\ skos : relatedTo\ T) \rightarrow (P\ hasTag\ U)$. Thanks to the support of negative tags, the antonym relationship can be leveraged too.

Structured tags allow to represent the same information in many different ways. This, however, is an advantage. It gives users freedom and allows to distinguish between possibly slightly different concepts in different communities. For example a community of photographers might use structured tags of the form "*geo:(x:NUMBER, y:NUMBER)*" to geo-tag their albums and photos. A community of travellers may use a different form: "*geo:NUMBER:NUMBER*" to tag their blog posts about travelling. Although it may seem as duplication at first, it may later turn out that the two communities use different coordinate systems. In that case, they will benefit from the difference in tagging and will be able to easily define rules to transform the coordinates to a common system.

5 Reasoning

Reasoning and reason maintenance are tightly related; they are the two main components that constitute a reason maintenance system. This section describes the proposed kind of reasoning with a focus on its Wiki-like specifics. The next section then describes the reason maintenance part of the system.

5.1 An approach to social automated reasoning

Reasoning should support the main defining traits of Wikis: simplicity and collaboration. Wikis are social, people work together in them and the KiWi platform is going to stress and support the social aspect even more. Therefore reasoning should also strive to enhance and simplify the collaborative aspect of work. This section discusses and proposes an approach to automated reasoning which supports users in their collaboration. Note that, although tags are often discussed in the following, a generalization to annotations may be possible or even necessary.

5.1.1 Shared understanding

During collaborative work it is important that every member knows what other members think about the subject in question. In a Wiki-like environment, collaboration takes place in and around content items. Users' understanding of a content item in KiWi can be expressed via annotations, for example tags. Therefore one could see the set of tags of an item as the *shared undersdanding* of the item of all users who tagged it. Inconsistencies within a shared understanding may be indicative of disagreements or misunderstandings of the item between the users. Such inconsistencies can be important for the future success and efficiency of the collaborative effort. Reasoning should therefore support their uncovering so that users can spot them and their origin easily and take an action to correct them.

Often it is also important to know who worked on what topic, who approved which content item or who agreed with or promised something. This kind of user-centered information can be accessible to some extent via a simple search function. There is however more to expect from an environment enriched with different kinds of annotations and reasoning. These advanced features open the door to a whole new area of useful questions about user-user and user-content interactions which would not be easily answerable otherwise. Examples of these questions include:

- What is the item that people expressed the most agreements with during the last week?
- What is the most controversial item if the degree of controversy is defined as $\frac{1}{|\frac{1}{\#agree}-\frac{1}{\#disagree}|}$?
- Which team causes the most inconsistencies in the system?
- What are the inconsistencies stemming from data entered by team X?
- What are the inconsistencies stemming from (user-)data originating in content items X, Y, and Z?
- What follows from information entered by John, Melissa, and Patrick? (shared understanding of John, Melissa, Patrick)

To be able to answer these questions the system has to

- support agreeing and disagreeing,
- be able to work in presence of inconsistencies and to detect them,
- track where which data originated and possibly also
- support user-dependent or content-dependent knowledge derivation.

Agreements and disagreements can be supported via tagging as discussed previously. Paraconsistency is a core feature of the reasoning under consideration, see Section 5.2 for details. The main focus of this section is on the last two points. Tracking of the origin of data is, to some extent, also the core feature of our reasoning. On the other hand, the above examples indicate that the interesting information about the origin of derived facts is not restricted only to the set of facts and rules on which the inferences were based; it is also interesting which user asserted the facts and where, in which content item.

Authorship of derived tags. One question that arises with reasoning about tags is who is the author of a derived tag. Consider the following example. There is a content item with two tags "*bug*" and "*–processed*" and a derived tag "*todo*" (derived via rule $R_1 : bug \wedge -processed \rightarrow todo$). Let us assume the "*bug*" tag was created by Melissa and the tag "*–processed*" by John and the rule by Alice. Who is the author of the "*todo*" tag? There are three possibilities and each has its merits. The author of a derived tag can be:

- the author of the rule
  - In a sense, it is the author of the rule who derived the new information and the authors of the source tags may even not be aware of it.
- the authors of the source tags
  - This option stresses the collaborative aspect of tagging and reasoning. It also may be the case that the rule was defined by an administrator or by an advanced user who otherwise does not have anything in common with the work and tagging itself.
- both the author of the rule and the source tag authors
  - They all worked together to produce the derived information.

Each of these options has its merits and which is the best depends on the user behaviour. The first option is the cheapest to implement but does not bring much interesting information. The second and third options allow to track the origin of derived information. Therefore we distinguish between the author of derived information which is the author of the rule and between the *user-origin* of a derived tag which are all users based on whose tags a new tag was derived. A system administrator can configure whether the user-origin of derived tags should include rule authors.

Consider the above example again. The author of the derived "*todo*" tag is the author of the above rule. The user-origin of the tag consists of both Melissa and John because the tag depends on tags by both of them. Let us assume that Patrick, a programmer, tags the content item as "*wontfix*" and there is a constraint rule $R_2 : todo \wedge wontfix \rightarrow \bot$ in the Wiki. Then, there is an inconsistency derived as a result of tag assignments of all the three people – the user-origin of the inconsistency.

5.1.2 Rule scope

In professional context the quality and source of information is often very important. For collaborative work it means that it can be effective only when people know with whom they collaborate. Consider the example from the previous section again. It could be the case that only the marketing department is allowed to decide which bugs must be fixed. Rule R_2 should therefore apply only to tags by users who work in the marketing department; the rule's *scope* should be limited only to marketing users.

Limiting the rule scope with respect to users is only one of several options. Rule scope restricts a rule to only certain data. Data in a Wiki are usually created by *users* and divided into *pages*. Therefore, content items are another candidate for a rule scope.

User scope and content item scope. Users and content items are the most important candidates for rule scope in KiWi. After all, almost everything in KiWi is a content item, including users. However, a distinction has to be made between users and content items with respect to scope. Restricting a rule to a set of users understood as a set of content items would restrict them to the actual content items and not to the tags entered by these users.

Let us summarize what are the advantages of rules with a rule scope. Rule scope

- allows to restrict a rule to only trusted users and/or content items,
- allow users to use different rules for the same tags in different content items (e.g. different communities),
- helps to modularize the rule base. Restricting a rule to certain data reduces the possibility of interaction with other rules. Therefore it helps to avoid and prevent unintended rule interactions and thereby it reduces the cognitive load associated with rule base development.

Other notions of scope. There could be other candidates for a rule scope, notably time - to restrict a rule to only a certain period of time. There is, however, a significant difference between time as scope and users or content items as scope. Users and content items are a kind of provenance data of derived annotations meaning that they change according to who and where created the annotations used in the rule antecedent. Time, in contrast, does not propagate via rules. This means that while a rule with a time scope can easily be rewritten to an equivalent rule without time scope, it is not as simple in the case of rules with a user or content item scope. Such rules still could possibly be rewritten but they would necessarily have to "walk the tag dependency tree" and gather all the (user and content item) information on the way. This would have to be done before each rule application for each tag. So, at the very least, it would be inefficient. Therefore a further support from the rule language and reasoning is necessary for efficient evaluation of rules with user and/or content item scopes. The focus in the following is therefore on user and content item scopes as they are more than a mere syntactic sugar.

Extensional vs. intensional scope. The notion of scope could be further generalized. So far, it was silently assumed that the user and content item scope consist of an explicit list of users and an explicit list of content items. In other words, it was assumed that the scope is an *extensionally* defined group of users or content items. The groups could also be defined *intensionally*: a rule could be meant to be applied only to tags by managers who have worked for the company for at least three years. The KiWi system will support extensional scopes. Intensional groups remain as a possible future extension. In the following text, the words *group* and *scope* refer to the extensional meaning unless stated otherwise. See the end of Section 5.1.1 for a further discussion of intensional groups.

Rule scope and negation as failure. Rule scope limits rule application to only certain data. Negation as failure in a rule is also subject to this limitation. See Figure 1. If a rule, such as rule R_1, with negation as failure has scope consisting of two users, Melissa and John, then a negated tag in the rule body will be satisfied even if there exists this tag but is assigned by someone else than Melissa or John. In the scenario in Figure 1, Patrick tags content item "Ci1" as "*reviewed*" which does not lead to removal of the tag "*toReview*" because Patrick is not in

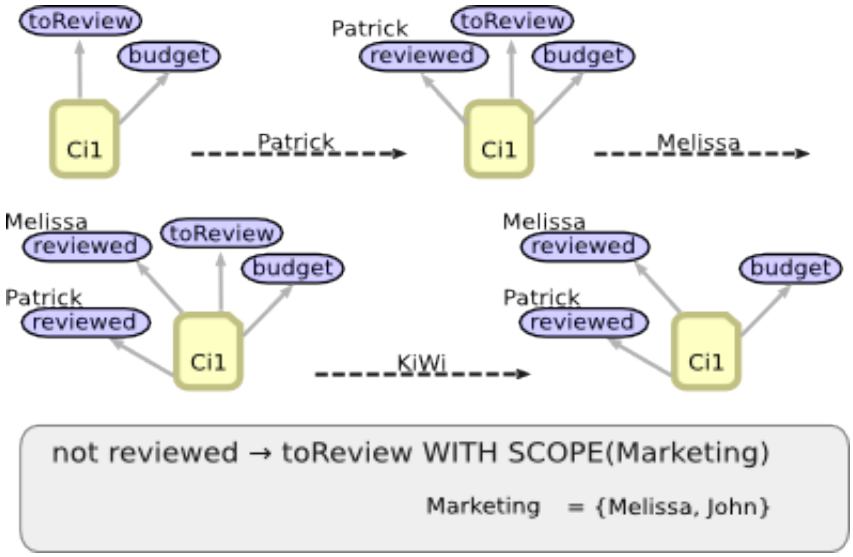

Figure 1: A scenario illustrating how rule scope affects negation as failure.

the scope of rule R_1. The "*toReview*" tag is removed only after Melissa tags the content item as "*reviewed*". Rule scope in combination with negation as failure therefore leads to a concept similar to Axel Polleres's scoped negation as failure [27].

"Switching off" rules. Rule scope affects what annotations are derived and therefore what the user may eventually see. A user may also wish to *not see* the effects of some rules and to do so without affecting other users. From one point of view, this may be seen as a task for personalization because it only changes the way the way the knowledge in the system is presented, not the actual knowledge. From another point of view, possibly many dependency subtrees of facts have to be hidden when a rule is "switched off". This means that the operation is far from trivial and would probably require support from a reasoner.

5.1.3 Computing shared understanding

This section explores how shared understandings described in Section 5.1.1 can be computed. To recapitulate, shared understanding of a group of users of a group of content items is the set of all tags of these content items assigned by these users. Knowing what a shared understanding looks like may help users to assess the influence a group of users or a group of content items has on the overall state of knowledge. Thus, shared understandings can play the role of additional explanation of the system behaviour. This section outlines how shared understandings can be computed in an efficient fashion.

Definition 1 *Author of an explicit tag is the user who assigned the tag. Author of an implicit tag is the user who created the rule by which the tag was derived.*

User-origin of a tag consists of all the users based on whose tags the tag was derived. As there can be multiple ways to derive a certain tag, the user-origin can consist of multiple sets

of users.

Definition 2 *User-origin of a tag is a set of sets of users. User-origin of an explicit tag t, denoted $UO(t)$, is a set containing a singleton set containing the author of the tag. User-origin of an implicit tag t derived via rule $B \rightarrow t$, where $t_1, ..., t_n$ are all tags in the rule body B, is*

$$UO(t) = \{\bigcup_{i=1}^{n} s_i | s_1 \in UO(t_1) \text{ and } \ldots \text{ and } s_n \in UO(t_n)\}$$

Consider the example from section 5.1.1: $UO(bug) = \{\{Melissa\}\}$, $UO(-processed) = \{\{John\}\}$. The user-origin of the "*todo*" tag derived by rule $bug \wedge -processed \rightarrow todo$ is then $UO(todo) = \{\{Melissa, John\}\}$. If, in addition, it was possible to derive the tags "*bug*" and "*-processed*" by other rules and taggings by John's and Melissa's colleagues Alice and Bob and the user-origins were $UO(bug) = \{\{Melissa\}, \{Alice, Bob\}\}$ and $UO(-processed) = \{\{John\}, \{Alice\}\}$ then the origin of "*todo*" would be

$$UO(todo) = \{\{Melissa, John\}, \{Melissa, Alice\}, \{Alice, Bob, John\}, \{Alice, Bob\}\}$$

Similarly for content-item-origins. Content-item-origin of a tag consists of all the content items based on tags of which the tag was derived. As there can be multiple ways to derive a certain tag, the content-item-origin can consist of multiple sets of content items.

Definition 3 *Content-item-origin of a tag is a set of sets of content items. Content-item-origin of an explicit tag t, denoted $CIO(t)$, is a set containing a singleton set containing the content item to which the tag was assigned. Content-item-origin of an implicit tag t derived via rule $B \rightarrow t$, where $t_1, ..., t_n$ are all tags in the rule body B, is*

$$CIO(t) = \{\bigcup_{i=1}^{n} s_i | s_1 \in CIO(t_1) \text{ and } \ldots \text{ and } s_n \in CIO(t_n)\}$$

Definition 4 *Shared understanding of a set of users U of a set of content items C, designated $SU(U, C)$, is*

$$SU(U, C) = \{\ t \mid t \in Tags(C) \text{ and } (\exists s \in UO(t)) s \subseteq U\}$$

where $Tags(C)$ is a set of tags assigned to content items C. It is a set of tags where the user-origin set of each tag contains a set which is a subset of U.

Definition 5 *User-scope of a rule is a set of users. Content-item-scope of a rule is a set of content items.*

Each rule has a user-scope and a content-item scope. The default user-scope is the set of all users. The default content-item scope is the set of all content items.

Looking at the above example by Definition 2, it is easy to see that tracking origins can lead to a combinatorial explosion if all possible combinations users and all possible combination of content items are tracked. For n users there are $n!$ possible user-origin sets. On the one hand, users can ask a question about any of these user-origin sets, on the other hand, it is likely that they are interested mainly in the sets that correspond for example to teams of their company. A team is basically a group of employees, in our case that would be a group of users. It is likely that teams would be defined as groups of users in an enterprise Wiki. Therefore, it is reasonable to focus on tracking origin-sets with respect to predefined groups of users and content items. This way the combinatorial explosion can be avoided while providing the same functionality to users.

User-origin sets with groups. When tracking user-origin sets using groups, the way of computing the origin sets is basically the same, only each origin set is replaced by the smallest predefined "group" which subsumes the origin set [6]. The user-origin set is then a set of predefined groups. For this to work correctly and as expected, a number of groups (from the full lattice of groups) has to be added. First, a group containing all users has to be added to the predefined groups so as to ensure that there always is a smallest encompassing group. Also, a group for each single user needs to be added as it is natural to require that user-scopes are possible to define for single users too.

With groups, the shared understanding of a set of users U can be computed the same way as in Definition 4 as long as the set of users U is one of the predefined groups. If the set of users is not one of the predefined groups then:

$$SU(U, C) = \bigcup_{i \in I} SU(G_i, C) \upharpoonright (G_i \cap U) \tag{1}$$

where $|G_i \cap U| \geq 1$ for all $i \in I$, G_i are all the predefined groups, and $T \upharpoonright G$ filters out all tags from the set of tags T the user origin-set of which does not contain a set which is a subset of G, and $DK(U) \upharpoonright \emptyset = \emptyset$. Equation 1 is correct under the assumption that each rule has one of the predefined groups as a user-scope (because then if U has a subset U' which is not subset of any G_i then there are no derivations dependent on tags by users U'). A consequence of this assumption is that the administrator of the system can influence what gets precomputed by creating predefined groups. The assumption could be generalized to allow the scope to be a set of groups. This would allow the user to restrict a rule to a combination of teams (resp. teams and employees) instead of restricting it to a single group.

Content-item origins with groups. Content-item origins can be adapted in a similar way to allow to work with groups of content-items. A group of content items could be imagined as a working space of a group of users which is separate from other content items with respect to annotations and other users. In this sense, each group of content items could be seen as a "knowledge space" - a closed group of content items "generating knowledge" which is self-contained: depends only on these and no other content items. If the KiWi system included a mechanism for dividing pages into strictly separate groups (e.g. with different access rights, etc.), the mechanism of a group of content-items as a rule scope would allow to ensure this separation at the level of annotations too.

Intensional groups. Intensional group is a group defined by a description of properties that members of the group must satisfy. In KiWi, such a description consists of a rule which assigns a tag to content items which satisfy the desired condition. For example, a group of senior managers can be created via a rule such as: $hasType\ kiwi : User \wedge manager \wedge hasExperience\ kiwi : SeniorExperience \rightarrow seniorManagerGroup$. The senior manager group then changes along with changes in annotations of kiwi user profile pages. A group can change in two ways: either a new member is added or an old member is removed. In the case of extensional groups, the system is informed of these changes immediately without the need to run a reasoner. Intensional groups, on the other hand, depend on results of reasoning which means that potentially any change in annotations can lead to changes in intensional groups. On the top of it, a change in an intensional group can lead to further changes in intensional

[6] This is correct if each rule scope is one of the predefined groups.

groups directly ("if a user belongs to a group of executives then he or she also belongs to the group of senior managers") or indirectly (groups are used as rule scopes and thus changes in groups affect what can be derived). This can lead to complex behaviour and with negation as failure even to reasoning loops: suppose there is a rule $senior \wedge not\ contractor \rightarrow eligible$ and a user, *Bob*, who initially is eligible. Let us further assume that there is a rule: $x \rightarrow contractor\ WITHSCOPE\ eligible$ then once user *Bob* annotates his user page with "x" then, because he is eligible, it is derived that he is a contractor which in turn leads to the conclusion that he is not eligible thanks to the first rule. This in turn leads to the removal of the contractor annotation because *Bob* is no longer in the scope of the second rule and thanks to the first rule his user page is annotated as eligible again, etc. Support of intensional groups in rule scopes can lead to a complex behaviour even in languages without negation as failure and thus counteracts rule scope as a means to limit the cognitive load while developing a rule base. Therefore KiWi will support explicitly defined groups and will leave intensional groups as a possible future extension.

5.2 The KiWi reasoning language

This section summarizes properties of the KiWi language used by the reasoner, a part of the work was published in [8]. At a later stage of the project, the reasoning language will be unified with the KWQL query language described in deliverable D 2.2 [9]. We identified the following requirements on reasoning for a semantic social platform. The language and reasoning have to:

- be intuitive and sufficiently easy to use in order to be widely accepted,
- be able to cope with formal (RDF/S) as well as semi-formal (structured tags) and informal (tags) annotations,
- work in presence of inconsistencies and support their discovery,
- be amenable to easy-to-grasp explanations of the system's deductions,
- support "partial knowledge reasoning" to convey the social dimension of social tagging,
- support negation as failure,
- be finitist - deductions yielding infinitely many implicit tags will hardly be required in social tagging.

The language. The KiWi reasoning language is a first-order logic language with connectives $\wedge$ and $\rightarrow$. The language further has two special formulas: $\top$ (read verum) and $\bot$ (read falsum) that can be seen as 0-ary connectives. Formulas and subformulas are defined as usual. There are three kinds of formulas: constructive rules, denial rules, and queries - all are assumed to be universally closed. Query is a well-formed formula built using the connectives $\wedge, not, \top, \bot$. Constructive rule is an expression of the form $Q \rightarrow A\ WITHSCOPE\ U, C$ where Q is a query, A is a atom (A cannot be neither $\top$ nor $\bot$) and U and C are user-scope and content-item-scope respectively. A denial rule, or constraint, is an expression of the form $Q \rightarrow \bot\ WITHSCOPE\ U, C$, where Q is a query, A is an atom, and U and C are user-scope and content-item-scope respectively. In addition, all rules are range restricted. This means that every variable occurring in A or in a subformula N of Q such that N has negative polarity

in Q also occurs in a subformula P of Q such that P has positive polarity in Q. Also, only stratified rule-sets are allowed to ensure good reasoning properties. This means that rule sets such as $\{not\, p \rightarrow p\}$ and $\{not\, p \rightarrow q, q \rightarrow p\}$ will not be allowed, see [6] for a formal definition of stratification.

Syntactic shortcuts. Because the language has to be user-friendly to casual users it is desirable that it supports syntactic constructs which simplify rule specification. It is assumed that non-expert users will use the language mainly to specify rules about tags (see Section 5.4 to learn more about the kinds of rules with respect to annotation type). Therefore the language supports an easy way to specify such rules. Namely, each conjunct which consists only of a constant or a variable is assumed to refer to a tagging of a page. Therefore a rule $A_1 \wedge \ldots \wedge A_n \rightarrow H$, where H and A_i are a constant or a variable, is equivalent to the rule $A_1 \wedge \ldots \wedge (C\ hasTagging\ T_1) \wedge (T_1\ hasTag\ A_i) \wedge \ldots \wedge A_n \rightarrow (\exists T_2)(C\ hasTagging\ T_2) \wedge (T_2\ hasTag\ H)$, where C and T_1 and T_2 are variables different from all other variables in the rule. This syntactic shortcut allows an easy specification of rules about tags. For example the rule $jaguar \rightarrow animal$ is equivalent to the rule $(C\ hasTagging\ T_1) \wedge (T_1\ hasTag\ jaguar) \rightarrow (\exists T_2)(C\ hasTagging\ T_2) \wedge (T_2\ hasTag\ animal)$.

It is likely that there will be support of additional shortcuts to allow for easy referring also to other user-level constructs described in Section 2.

5.3 Standard reasoning with RDF/S

This section presents a short overview of RDF/S semantics and how it is implemented in current systems.

5.3.1 RDF/S semantics overview

RDF is a language for knowledge representation, access, and use over the World Wide Web. RDF Schema (RDF/S) is a vocabulary based on RDF for the purpose of creating basic ontologies. The formal semantics of RDF and RDF Schema is defined in a W3C specification [18]. The document also provides a list of axiomatic triples which describe the nature of RDF and RDF Schema primitives and a list of entailment rules which express what triple is to be added to a graph that contains triples matching a pattern according to the RDF/S model theory. Following is the list of RDF/S rules (rules handling literals and blank nodes are excluded):

RDF1 (X A Y) → (A rdf:type rdf:Property)

RDFS2 (X rdfs:domain Y) ∧ (U X Z) → (U rdf:type Y)

RDFS3 (X rdfs:range Y) ∧ (U X Z) → (Z rdf:type Y)

RDFS4a (X A Y) → (X rdf:type rdfs:Resource)

RDFS4b (X A Y) → (Y rdf:type rdfs:Resource)

RDFS5 (X rdfs:subPropertyOf Y) ∧ (Y rdfs:subPropertyOf Z) → (X rdfs:subPropertyOf Z)

RDFS6 (X rdf:type rdf:Property) → (X rdfs:subPropertyOf X)

RDFS7 (X rdfs:subPropertyOf Y) ∧ (U X Z) → (U Y Z)

RDFS8 (X rdf:type rdfs:Class) → (X rdfs:subClassOf rdfs:Resource)

RDFS9 (X rdfs:subClassOf Y) ∧ (A rdf:type X) → (A rdf:type Y)

RDFS10 (X rdf:type rdfs:Class) → (X rdfs:subClassOf X)

RDFS11 (X rdfs:subClassOf Y) ∧ (Y rdfs:subClassOf Z) → (X rdfs:subClassOf Z)

RDFS12 (X rdf:type rdfs:ContainerMembershipProperty) → (X rdfs:subPropertyOf rdfs:member)

RDFS13 (X rdf:type rdfs:Datatype) → (X rdfs:subClassOf rdfs:Literal)

Most systems use these rules and forward or backward chaining (or their combination) to derive new facts and answer queries.

5.3.2 Reasoning techniques

There are many approaches to RDF/S reasoning but they can be divided into two categories: either they perform RDF/S reasoning directly or they transform RDF/S rules into some other formalism. Direct reasoning approaches are based on forward and backward chaining of RDF/S entailment rules, their combination, and often RDF/S-specific optimizations. Indirect reasoning is performed by translation of RDF/S rules and facts into another formalism such as conceptual graphs [29], F-logic [19], or SQL. The actual reasoning is then performed inside the destination formalism.

5.3.3 Direct RDF/S reasoning

Forward-chaining. Forward-chaining, also called materialization, is a rule-based reasoning method which given a set of rules and a set of base facts computes their deductive closure – it "materializes" all (implicit) facts which are entailed by the set of rules and the set of base facts. Forward-chaining is usually invoked upon each knowledge-base update so that implicit facts are always materialized so that answering a query then amounts to a simple fact lookup. See [6] for a formal description of forward-chaining.

Backward-chaining. Backward-chaining is a rule-based reasoning method which works backwards from a query (a goal) and tries to find a path from the goal to base facts via the rules. In logic programming, backward-chaining is implemented via resolution, most commonly SLD (Selective Linear Definite clause) resolution or its extension the OLDT resolution. Interestingly, backward-chaining can be used to evaluate rules in the forward-chaining manner. The method, called Backward-Fixpoint-Procedure (BFP), uses a meta-interpreter evaluated in forward-chaining manner to evaluate object-level rules in the backward-chaining manner. Again, for a detailed description of these techniques see [6].

5.3.4 Optimization techniques

Practical RDF/S reasoners do not rely only on one reasoning technique but combine them and also apply various optimizations. The choice between different reasoning techniques often constitutes a trade-off between time and space complexity. Backward and forward chaining can be combined so that a part of the knowledge base is materialized at update time and the rest

is then evaluated via backward-chaining at query time. This way, a smaller storage space is necessary while maintaining a reasonably fast query evaluation.

RDF/S specific optimization techniques include elimination of redundant derivation using entailment rules dependency analysis, query rewriting, and so called "labelling schemes". Broekstra and Kampman [4] analyse dependencies between RDF/S entailment rules to avoid re-derivation of already derived facts during forward-chaining. Stuckendschmidt and Broekstra [30] present a method to compute only such a part of all the implied statements that the rest of the implied statements can be computed by simple query rewriting at query time. Labeling schemes, described in [10], can help to optimize the implementation of transitive properties, particularly rdfs:subClassOf and rdfs:subPropertyOf, by assigning and processing special labels to nodes of the RDF/S graph.

5.3.5 Indirect RDF/S reasoning

RDF/S entailment can be performed also indirectly, after translation to another formalism. For example, Bruijn and Heymans [13] describe various kinds of RDF entailment in F-Logic and especially the Datalog fragment of F-Logic which allows to take advantage of optimization techniques for deductive databases. Another approach is taken by Oracle [31] who implement RDF/S (and partial OWL) reasoning by translation of rules to SQL. Corby et al. [12, 11] describe a method of RDF/S inference which works by translating the RDF/S graph to a Conceptual Graphs model; they use it as part of the Corese search engine.

5.4 Reasoning and transactions

[7] Interactivity is one of important features of a user-friendly software. Even rule-based reasoning and knowledge base updates could take long time because of long chains of dependencies in the knowledge base which could in turn hinder the responsiveness of the system. One approach to overcome this problem can be to queue reasoning requests and to run them in a separate thread so that the system is responsive but maybe not with fully up to date inferences. A complementary approach can be to try to leverage the nature of rule-based reasoning and the character of RDF/S to split a possibly long reasoning transaction into more shorter ones. To illustrate the point, consider the following scenario, see Figure 2.

There is a page tagged with two structured tags: "*type:itProject*" and "*manager:alice*". The knowledge base initially contains four triples: two from a use case ontology "UCONT" which say that each SwProject is a Project and each ITManager is a Manager and two (simplified) triples for the two tags. In addition, there are five rules: four user-defined and one from RDF/S model theory. These rules are used by the reasoner to infer new information. Figure 3 shows the new state of the knowledge base - five new triples were derived.

While the reasoner infers new facts, its reason maintenance component creates a justification dependency graph for these facts, see Figure 4.

First, to realize that a reason maintenance update is not always trivial, let us assume that the "*type:itProject*" tag is removed. It means that e.g. the left justification of fact $f8$ is not valid anymore. Therefore a naïve reason maintenance algorithm could check whether there is another valid justification of this fact. It could find the justification on the right side, $(f5, f7, u4)$, which is still marked as valid at this point. In this example, fact $f8$ would eventually get invalidated

[7] This section is based on work in cooperation with the Salzburg Research partner and was presented at ESWC09 poster session.

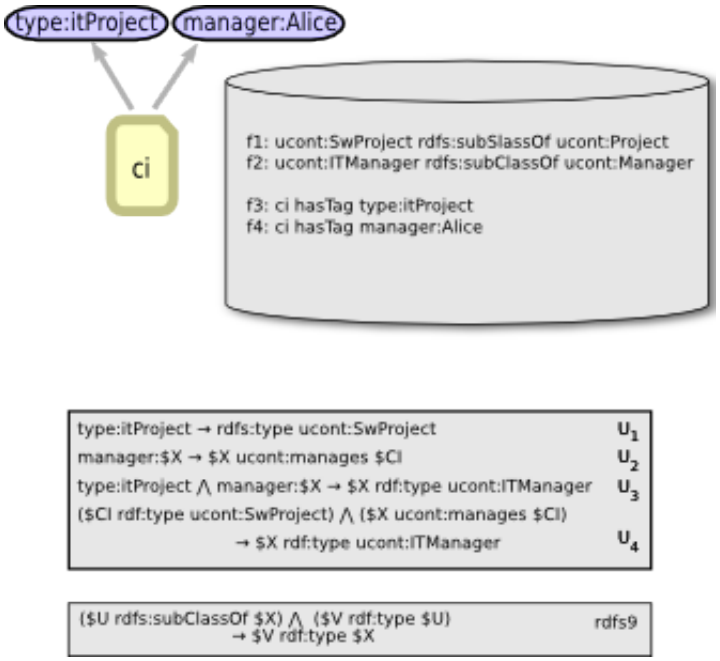

Figure 2: Initial state of knowledge base and page taggings together with rules to derive new facts.

even using this naïve algorithm. There, however, are cases where justifications are cyclic and where this algorithm would fail. Therefore the correct algorithm has to work in two steps: first it walks all justifications dependent on the removed fact, invalidates them and marks their consequences and propagates in the same way further. In the second step, it goes through all the marked facts and checks whether a well-founded support[8] can be found for them. The same mechanism may be triggered if a fact is added which is refered to by a rule in a negated form (negation as failure). This illustrates that even a simple change could take unexpectedly (from the user point of view) long time to be processed.

Reasoning transactions could be split into several smaller ones. It can be noticed that there are four types of rules with respect to the type of annotations they operate with:

- Rules about tags.
- Rules mapping tags to RDF/S annotations.
- Rules about RDF/S annotations.
- Rules mapping RDF/S annotations to tags.

The first and last type of rules derive information that is visible to users as tags. The second and third kind of rules infers RDF/S facts which may or may not immediately affect how the system is perceived by users. Therefore, reasoning could first process rules about tags so that users see changes in taggings as soon as possible and deal with the other kinds of rules in a later

[8]Refer to deliverable D2.3 for details about well-founded support.

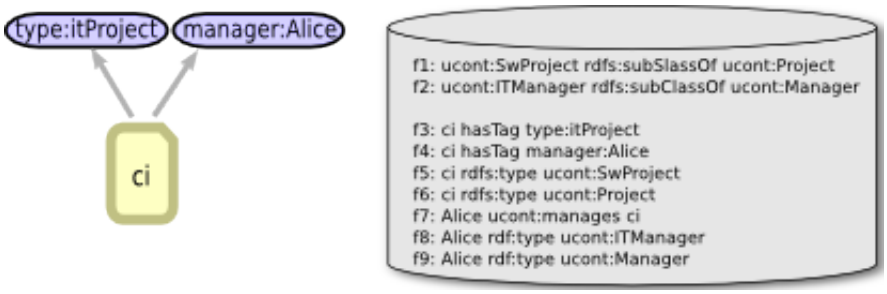

Figure 3: State of the knowledge base after the reasoner is run.

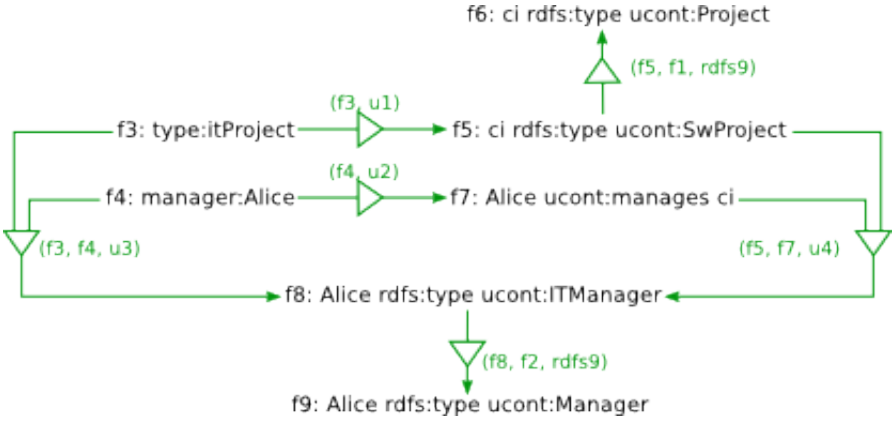

Figure 4: The justification dependency graph created by the reason maintenance component of the reasoner. Justifications are green.

transaction. Of course, there can be interdependencies between these kinds of rules which can make their completely separate processing impossible or difficult. It still can prove effective in some cases and will be subject of further investigation.

6 Reason maintenance

Reason maintenance was selected as a technique which will allow to perform knowledge base updates efficiently and to compile explanations for the system's deductions. Reason maintenance was previously applied to inferencing with RDF Schema by Broekstra [4] where the reasoner is tailored specifically to monotonic RDF/S reasoning. That is, it relies on RDF/S reasoning being monotonic and leverages dependencies among RDF/S model theory entailment rules to make non-redundant (and therefore faster) forward-chaining inferences. It uses backward-chaining to derive justifications for each statement. It cannot use the forward-chaining inference to record all justifications because of the optimization and for reason maintenance it is necessary to know all possible justifications. The setting of reasoning in KiWi is different in two primary aspects: it has to support user-defined rules as well as RDF/S model theory entailment rules and it should support negation as failure (i.e. non-monotonic reasoning). Also, Broekstra claims to build

upon the original Jon Doyle's reason maintenance system which is non-monotonic. There have been other reason maintenance algorithms proposed specifically for monotonic reason maintenance as well as other specific conditions (such as a fast and efficient reason maintenance system for embedded systems), see the state of the art report about reason maintenance in deliverable D2.3 [7]. Reason maintenance in KiWi will draw from the results of the survey too.

6.1 Incremental reasoning and processing updates

The naïve approach to knowledge base updates recomputes the full deductive closure from scratch upon a removal. Reason maintenance helps to avoid the need to recompute the full deductive closure because it keeps track of justifications, i.e. reasons why each fact was derived. Thanks to justifications, the reasoning and reason maintenance components can process updates without having to examine the whole knowledge base; they can focus only on the affected parts. This in turn enables incremental forward-chaining reasoning.

6.2 Theory

Work has been started to develop a declarative description of the KiWi reason maintenance system. Reason maintenance systems consist of two components: the reasoner and the reason maintenance component. The reasoner derives new deductions and the reason maintenance component records them and the reasons why they were derived. Reason maintenance algorithms then use justifications to process updates. In a sense, the reason maintenance component carries out a special kind of reasoning. This observation was exploited to devise a unifying logical framework for reason maintenance [14] - the work unifies description of all different reason maintenance systems within the framework of a special logic due to Dov Gabbay. The value of the work lies in the fact that it allows to describe reason maintenance systems in a unified way and therefore compare and analyse them more easily and to extend them. Similarly, a unified description of the reasoner and the reason maintenance system would enable easier analysis of the sytem's behaviour and perhaps even a simpler implementation in a sense. To this end, work has been started to develop a declarative description of a reason maintenance algorithm using the the KiWi reasoning language itself. If there was such a description it would be theoretically possible to use one reasoner (i.e. the reasoner implementing the KiWi reasoning language) to do the work of the both components within the reason maintenance system. To this end, it would be necessary to equip the KiWi reasoning language with a support of data structures and operators to manipulate them. Note, that these data structures can build on the concept of structured tags introduced in 2.2.2.

6.3 Explanations

The approach to explanation in KiWi is manifold and inherent in the general proposal. Explanation is a desired feature because behaviour of a system enhanced with reasoning can be complex and opaque. It can be seen as a remedy to fathom the system. Therefore explanation is only an addition to the general approach to simplicity and user-friendliness which comprises:

- support of informal and semi-formal annotations to make rules accessible to casual users,
- detection of inconsistencies,
- fact and inconsistency origin tracking,

- rule scopes to limit the cognitive load while constructing (and understanding) a rule base,
- explanation of derived facts and inconsistencies,
- interactivity - explanations should be interactive and quickly accessible, feedback already during rule development.

Therefore our approach is also in line with the four principles for building of better tool support for the creation of rule bases identified by Zacharias in [32]:

- Interactivity - tools should be interactive and support interactive try-and-error process of rule base creation,
- Visibility - users should be informed about possible rule interactions,
- Declarativity - all aspects or rule-base development should be declarative, i.e. including "debugging",
- Modularization - prevention of unintended interaction by providing means to modularize the rule base.

Explanations in KiWi will use justifications maintained by the reason maintenance component for an interactive presentation of fact derivation trees. The presentation will be enriched by information about user- and content-item-origins of facts and inconsistencies.

7 Reasoner implementation - initial plan and extent.

There are many features of reasoning described and proposed in this document. Because KiWi is an application oriented project and it is desirable that it is developed incrementally we propose a simplified reasoner which can be implemented faster than the whole proposal and which can be gradually extended later.

The core of the proposed reasoning and reason maintenance technology can be summarized as "a simple incremental inconsistency tolerant rule-based reasoning system which works both on formal and informal annotations, includes origin (or provenance) tracking, and allows for explanations". Therefore the simplified reasoner will support:

- rules about simple tags (both negative and positive),
- simplified, user-friendly syntax,
- inconsistency tolerant reasoning,
- origin tracking,
- user- and content-item-scope with explicit groups,
- basic RDF reasoning,
- updates based on a reason maintenance algorithm,
- simple explanations.

The reasoner will be implemented using the forward-chaining reasoning strategy as it poses less design problems than backward-chaining or a mixture of the two. The following features remain as possible future extensions: negation as failure (and the respective reason maintenance algorithm), aggregation, structured tags, better RDF/S support, arithmetics.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no 211932. See also http://www.kiwi-project.eu/.

References

[1] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, volume 2006, 2006.

[2] Tim Berners-Lee. Linked data. *W3C Design Issues*, 2006.

[3] Dan Brickley and R.V. Guha. Rdf vocabulary description language 1.0: Rdf schema.

[4] Jeen Broekstra and Arjohn Kampman. Inferencing and truth maintenance in rdf schema - exploring a naive practical approach. *Workshop on Practical and Scalable Semantic Systems (PSSS)*, 2003.

[5] François Bry, Michael Eckert, Jakub Kotowski, and Klara Weiand. What the user interacts with: Reflections on conceptual models for sematic wikis. 2009.

[6] François Bry, Norbert Eisinger, Thomas Eiter, Tim Furche, Georg Gottlob, Clemens Ley, Benedikt Linse, Reinhard Pichler, and Fang Wei. Foundations of rule-based query answering. In *Reasoning Web, Third International Summer School 2007*, volume 4636 of *LNCS*. Springer-Verlag, 2007.

[7] François Bry and Jakub Kotowski. Reason maintenance- state of the art. research report, PMS-FB-2008-17 PMS-FB-2008-17, Institute for Informatics, University of Munich, 2008. KIWI Deliverable.

[8] François Bry and Jakub Kotowski. Towards reasoning and explanations for social tagging. *Proc. of ExaCt2008 - ECAI2008 Workshop on Explanation-aware Computing. Patras, Greece*, http://www.pms.ifi.lmu.de/publikationen#PMS-FB-2008-2, 2008.

[9] François Bry and Klara Weiand. Reasoning and querying - concept and model. 2009.

[10] Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, and Sotirios Tourtounis. On labeling schemes for the semantic web. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 544–555, New York, NY, USA, 2003. ACM.

[11] O. Corby, R. Dieng-Kuntz, and C. Faron-Zucker. Querying the semantic web with corese search engine. In *ECAI*, volume 16, page 705, 2004.

[12] O. Corby and C. Faron-Zucker. Corese: A corporate semantic web engine. In *Proc. of the WWW'2002 Workshop on Real World RDF and Semantic Web Applications*, 2002.

[13] Jos de Bruijn and Stijn Heymans. Rdf and logic: Reasoning and extension. *Proceedings of the 18th International Conference on Database and Expert Systems Applications (DEXA 2007)*, page 5, 2007.

[14] Detlef Fehrer. A unifying logical framework for reason maintenance. In *ECSQARU '93: Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 113–120, London, UK, 1993. Springer-Verlag.

[15] David M. Geil. Collaborative reasoning: Evidence for collective rationality. *Thinking & Reasoning*, 4(3):231–248, 1998.

[16] T.R. Gruber et al. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5:199–199, 1993.

[17] Mandy Haggith. Disagreement in creative problem solving. Technical report, Department of Artificial Intelligence, University of Edinburgh, 1993.

[18] P. Hayes. Rdf model theory.

[19] Michael Kifer and Georg Lausen. F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 134–146, New York, NY, USA, 1989. ACM.

[20] T. Kvan. Designing collaborative environments for strategic knowledge in design. *Knowledge-Based Systems*, 13(6):429–438, November 2000.

[21] Frank Manola and Eric Miller. Rdf primer.

[22] A. P. Mcafee. Enterprise 2.0: the dawn of emergent collaboration. *Engineering Management Review, IEEE*, 34(3):38, 2006.

[23] T.H. Nelson. *Literary Machines*. Mindful Press, 1993.

[24] C.K. Ogden, I.A. Richards, B. Malinowski, and F.G. Crookshank. *The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism*. Harcourt, Brace & Company, 1938.

[25] E. Oren. Semantic Wikis for Knowledge Workers. 2005.

[26] C.S. Peirce. On a new list of categories. In *Proceedings of the American Academy of Arts and Sciences*, volume 7, pages 287–298, 1868.

[27] Axel Polleres, Cristina Feier, and Andreas Harth. Rules with contextually scoped negation. pages 332–347. 2006.

[28] F. Saussure. Course in general linguistics (W. Baskin, Trans.). *New York: Philosophical Library*, 1916.

[29] John F. Sowa. Conceptual graphs summary. pages 3–51, 1992.

[30] Heiner Stuckenschmidt and Jeen Broekstra. Time - space trade-offs in scaling up rdf schema reasoning. *WISE Workshops, Springer*, page 10, 2005.

[31] Zhe Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an inference engine for rdfs/owl constructs and user-defined rules in oracle. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1239–1248, 2008.

[32] V. Zacharias. Tackling the Debugging Challenge of Rule Based Systems. In *Enterprise Information Systems: 10th International Conference, Iceis 2008, Barcelona, Spain, June 12-16, 2008, Revised Selected Papers*, page 144. Springer, 2009.