ABSTRACT
Semantic wikis enhance normal wikis with typed links. Every link on a semantic wiki page has a type describing the relation of the current page to the target of the link. In this paper we propose an architecture that can be used to validate the user input and consistency of typed links in the input with the existing knowledge base. The validation of the consistency is done by utilising a reasoner component embedded in the architecture.

1. INTRODUCTION
For those among you that haven’t met a wiki yet, a wiki is a web-based service that enables users to collectively edit or create content stored on the host of the web-based service. One of the strong points of the wiki concept is the ease with which you can create links between the pages that are stored on the wiki.

Some wikis allow any visitor to make changes to the content and although this policy leaves the door wide open for vandalism the amount of wiki vandalism is quite low due to the ease of restoring older versions of the content. Other wikis can be set up to require the user to register in some way [Dok07], thus creating a barrier of sorts against one-shot vandalism.

Since the original wiki implementation of Ward Cunningham in 1994 a lot of different implementations have appeared, all focusing on slightly different designs and ideas. Two examples of this are MediaWiki [Med07], on which the well-known Wikipedia [Wik07] is built, and DokuWiki [Dok07], a small wiki designed with ease of installation in mind.

1.1 Semantic wikis
A relatively new development in the field of wiki implementations is that of the semantic wiki. This development is, of course, partially triggered by the current trend of the semantic web. In the semantic wiki all links between subjects are extended with a type. The trick is that every link between two subjects now has an extra notation describing what kind of link it is.

Take, for example, a wiki about cooking recipes: We have a lot of pages that describe ingredients and a lot of pages that describe the recipes itself, of course we also have pages about the chefs who invented the recipes. In a normal wiki the links between these pages would be just that; links (see Figure 1). However in a semantic wiki the links between recipe and ingredient would be typed with "ingredient", and the links between cook and recipe would be typed "invented" (see Figure 2).

These typed links bring quite a few benefits with them and they also bring in certain drawbacks. First we shall discuss the two benefits of typed links, thereafter we shall discuss the drawback.

A big benefit of the typed links are the enhanced search capabilities of the wiki, not only can you search for the term "tomato", you can have the wiki return all pages about cooks who use this "tomato" in the recipes they invented. You can also search all ingredients ever used by your favourite cook; there will of course be no guarantee that a combination of all these ingredients will taste well.
A second benefit is the automatic generation of a Resource Description Framework [LS04] description of the contents of the wiki, or parts thereof. This enables data aggregation tools and other interesting semantic web applications to use the wiki as a data source. A multitude of examples about semantic applications are available, ranging from PDA that assist the user with finding a nearby General Practitioner [BLHL01] to the plea of help from an agent-enabled fishing vessel that’s in bad weather [Hen01].

There is one strong drawback to the whole typed link idea. As with all things in a wiki, the user edits them. And since most users don’t have a clue about typed links at the moment the whole part about benefits is a bit of a moot point. This is true for every new technology that has not become a mainstream product yet. There is a good chance that one day the every-day wiki user will not only write down a new recipe, but that she will also create the typed links that hold the recipe and the ingredients together.

The typed links in a semantic wiki do not have any meaning on their own. To give the different types meaning an ontology is needed.

1.2 Ontologies
Simply put an ontology is an agreement to use the same terms in the same area of expertise. Examine the running example of the CookWiki, what stops a user from typing a link with "discovered" instead of "invented"? The ontology, whether it is written down or just assumed, confers an agreement on terms and relations between the subjects in some field of knowledge.

Ontologies can be described in formal languages such as OWL [BvHH+04], or they can be written down in a more informal style. Either way, ontologies enable users to communicate about subjects without ambiguity about the terms or relations they use in describing the subject. If a formal language is used, this ability to communicate extends to automated processes. One kind of automated process that is enabled by the formalisation of the ontology is automated reasoning.

Automated reasoning can be used to deduct facts from a knowledgebase. These facts are already in the knowledgebase in an implicit form, but the reasoning process, cunningly called the reasoner, brings them out as explicit facts qualified with a proof. Another use of the reasoner is checking whether adding an extra fact keeps the knowledgebase consistent.

The usefulness of the reasoning process can be seen if you think of the semantic wiki as one giant knowledgebase. This way the reasoner can be used to deduce information from the available subjects in the wiki.

2. PREVIOUS WORK
Quite some architectures have been proposed for the semantic wiki, they differ in the same way different wiki architectures differ. Some of the architectures focus on the structure that can be provided through the use of elaborate ontologies, i.e., KawaWiki [KKT06]. Others focus on integrating the typed links within the normal wiki text content, examples of this are SemperWiki [OBD06] and SHAWN [Aum05]. One interesting development is OntoWiki by Martin Hepp, Denial Bachlechner and Katharina Siopraes [HBS06], they proposed a wiki based system for editing ontologies.

To make sense of all the semantic information a reasoner is needed, one of those is Pellet [SP04]. Because the inner workings of the reasoner are beyond the scope of this paper we will not describe all the initiatives that have been taken up to now. Suffice to say that the field of automated reasoning is quite advanced in comparison to the field of semantic wikis. One paper that deserves special mention, because of its vision, is Reasoning in Semantic Wikis by Markus Krötzsch, Sebastian Schaffert, and Denny Vrandečić [KSV07]. Although the validation and checking of wiki content could alleviate some of the problems plaguing the wiki community very little work has been done in this direction. One of the few papers about validation in wikis is Constrained Wiki: an Oxymoron? [IZ06]. The paper describes an architectural extension that adds the concept of validators to the normal wiki work flow. These validators all validate a small part of the wiki page that is to be saved, if they find anything unsatisfactory they warn the user and won’t allow the page to be saved.

3. RESEARCH
The previous section shows that a lot of work has been done in the field of reasoning and the field semantic wikis. Unfortunately hardly any research has been done combining both fields. A lot of the semantics research tends to be of good use to the academics using the technology and most of the semantic wikis described in the references [Aum05, KKT06] are not in use by end-users.

Because of this it is of importance that online applications that benefit the end-user are researched. Bigger benefits from online applications enjoyed by end-users result in a greater motivation to enhance other systems, and pages, with semantics, thus joining bigger parts of the Internet to the semantic web.

3.1 Research Method
The research method used in this paper is based on the principles of software engineering. An analysis is made of the problem at hand and a set of requirements is produced by which the design is constrained. The requirements that must be satisfied by the architecture are set by analysing use cases. These use cases describe scenarios in which the end-user wants to get something done.

The use cases form the guide for the rest of the research. The extension to the architecture is based on these use cases, because the requirements follow directly from the use-cases.

3.2 Problem Statement
All good research is based on a, sometimes implicitly, defined problem statement. The problem statement that forms the basis for this research is as follows:

How to validate the input of users of a semantic wiki and how to use the feedback given by this process be used in such a way that the feedback does not interfere with the freedom of the user, as enjoyed in a more traditional wiki?

To focus the research we’ll propose an extension based on one of the architectures designed in previous work. To further limit the research we won’t look into the different kinds of automated reasoning and assume a reasoner is available.

4. USE CASES
The design of the architecture will be guided by four use cases. These use cases describe the processes that the end-user will partic-
The first two use cases are based on Reasoning in Semantic Wikis by M. Krötzsch, S. Schaffert and D. Vrandečić [KSV07]. Although wiki vandalism has been shown to be reverted within minutes [VWD04] it is still a point of improvement, the third use case describes yet another way wiki vandalism can be limited.

The last use-case is a necessary evil. Because an ontology is needed by the reasoner this ontology has to be available. The last use case describes the input of said ontology.

The use cases are:

- Consistency checking
- Simplifying input
- Countering Wiki vandalism
- Input of the ontology

The first use case, consistency checking, is exactly about what the name implies. In a full-fledged semantic wiki the user’s input will be checked for consistency before being added to the knowledge-base. This also implies that the ‘save’ action which can be used while editing a page is extended with a validation step.

Imagine, for example, a wiki in which we record information about countries and their capitals. If the knowledgebase already contains the capital “Amsterdam” for the country “The Netherlands”, then a user trying to edit the page of The Hague and, by accident, types the link to the Netherlands with a “capital” type this error will be caught during the consistency check.

The next use case, simplifying input, is about the assistance of the user while he or she is editing a page. When a user enters the type of a link the system can detect which existing pages fit all the constraints given in the ontology for that type of link. The system can then report all the available pages to the user, thus assisting him or her in editing the page by offering alternatives to choose from.

The same example as for the previous use case can be used here. While the user is editing the page of Amsterdam, and enters a “capitalOf” typed link, the system can then examine all pages, and offer those of which Amsterdam is the capital, “The Netherlands” in this case.

Countering wiki vandalism is always a good thing to do because it keeps all kinds of rubbish, like advertisements, away from the wiki. This use-case isn’t a use-case from the perspective of the end-user, it uses the perspective of the wiki administrator.

The input of the ontology is the last use case considered. The ontology should be entered in the wiki because without an ontology the validation scheme won’t work. The end-users can just edit the ontology as any other part of the wiki, thus enabling them to come up with the best ontology for their particular wiki in a collaborative way.

5. ARCHITECTURE

Because we want to extend the architecture of the semantic wiki we have to ask the following question: What is the architecture of the semantic wiki? To answer this question one must first know what the architecture of the normal wiki is.

5.1 Existing architecture

Almost all normal wikis [Wik07, Dok07] use a very simple architecture (see figure 3), in which there are three parts: the user interface, the business logic and the page storage.

![Figure 3: Normal wiki architecture](image)

The user interface handles all interaction with the user, and delivers feedback from the logic to the user. The logic itself handles all commands and queries the user orders through the user interface. The page storage stores all wiki pages in some way, be it a database or on the file system.

There is no such thing as ‘The Semantic Wiki Architecture’. Because of this we shall define a generic architecture based on implementation references.

Based on the architecture mostly used by normal wikis, which only contains storage for texts, and the architecture described by Völkel, Krötzsch et al. in their paper about Semantic Wikipedia [VKV06] it appears that the storage of semantic data is separate from the storage of the pages itself. This conclusion is strengthened by the fact that it is easier to implement the semantic storage apart from the page storage.

However, SemperWiki [OBD06] uses a different approach. The SemperWiki storage architecture consists of only one storage element which stores both text and semantic data. This approach is shared with SHAWN [Aum05]. Shawn also stores the text of a page in the same storage as the semantic data. Both of these systems store their page text as a single entry in the semantic storage.

We will assume that the generic semantic wiki architecture uses the separate storage architecture. This assumption is based on two things: the single storage architecture can easily be adapted to the separate storage and it is easier to add the separate semantic storage to an existing wiki. The second argument is built on the basis that it is more useful to extend existing wikis instead of re-inventing the wiki.

A short overview of the generic wiki architecture used in this paper is in order now that it has been defined. As can be seen in Figure 4 the semantic wiki has four distinct components. The first three components correspond to components seen in the normal wiki architecture. The new component is the semantic storage, this storage is used to store all semantic knowledge contained within the wiki pages. Every relation is represented by a triple containing subject, predicate and object. An extension to this triple is used to identify the wiki page on which it is defined, the source.

![Figure 4: Semantic wiki architecture](image)
architecture follows. The users has entered the following text at the fictional wiki page located at http://wiki.recipes.edu/Tomato:

==== Tomato ====
A tomato is a [[isA|Vegetable]].

The user input contains one typed link, a link to the page Vegetable with the type isA. The text will be processed by the logic, which will split the text in two components: The text itself, and the semantic data.

The text that will be stored in the text storage is a verbatim copy of the user input. One triple will be added to the semantic storage, in the form of (subject, predicate, object, source): (Tomato, isA, Vegetable, http://wiki.recipes.edu/Tomato).

5.2 Extended architecture
To enable validation based on anything the generic architecture needs to be extended with a validator component. This validator will become a key component in the extended architecture because almost every operation on the wiki or a page thereof needs to be checked by the validator. To enable validation based on reasoning the validator itself will contain a reasoning engine. Deciding on a specific reasoning engine is beyond the scope of this paper.

The reasoner needs one or more ontologies to validate the triples contained in a page. Because most ontologies are either based directly on a formal language or have a representation in text, the ontologies can be stored in the page storage like any other wiki page. This has the added benefit that the ontologies are available for editing the same way a wiki page is edited.

The validator component is added to the generic architecture on the same level as the logic. As can be seen in figure 5 the validator needs connections to the logic, the page storage and the semantic storage. The connection to the logic is obviously needed because data will be passed between the two components. The connection to the semantic storage is needed to procure the semantic information needed in the reasoning process. The validator has a connection to the page storage because it needs to procure the ontologies from the page storage for the same reasoning process.

5.2.1 Consistency Checking
The first use case, consistency checking, is the simplest of the three use cases. Figure 6 shows the flow of data during this use case. The user inputs a new page, or edits a page (1). This page is then passed to the validator (2), which uses its reasoning engine and the data from the semantic storage (3a) in combination with the ontology (3b) to determine whether the page is consistent with the known data or not. The information from the semantic storage will be filtered to clean out any semantic information that was defined in the old version of the changed wiki page.

If the page is found to be consistent (4) with the current semantic information the logic stores the page in the page storage (5) and overwrites any triples in the semantic storage originating from the page with the new triples contained in the page (6). Afterwards the user is informed about the successful addition to the wiki (7).

If the page is found to be inconsistent with the current knowledge the page will not be stored (number 5 and 6), and the inconsistencies will be reported to the user (7).

5.2.2 Assisting with input
The second use case concerns assisting with user input. While the user enters the start of a typed link in the contents of a new page or while editing the contents of an existing page (1) the logic queries the validator for a list of candidate pages that can be used to complete the link (2). The validator again uses the reasoning engine in combination with both the ontologies (3) and the semantic information (4) to deliver a list of candidates to the logic (5). The logic then passes this list back to the user (6) who can then use the list as an aid to decide to which pages she can link. The data flow of this operation can be seen in figure 7.

This use case can only be implemented if the user interface supports it. A discussion about the implications of user interface feedback are beyond the scope of this paper.

5.2.3 Countering Wiki vandalism
The third use-case, countering wiki vandalism, is from the perspective of the wiki administrator. When the wiki administrator sets up the wiki so that it only accepts typed links, some types of vandalism will either be very difficult to pull off, or easier to track.

Some relations in the ontology will be more susceptible to vandalism and although they might allow a wide range of different links,
most of these links will be suspect because of the strange combinations of concepts. A good example of this is the "partOf" relation, "Amsterdam" has a "partOf" relation with "The Netherlands", but it shouldn’t have a "partOf" relation with "Belgium".

5.2.4 Input of the Ontology
The last use case, pertaining the input of the ontology, follows the same general pattern as the first use case. However, because the user edits the ontology, the very basis on which the whole reasoning process is built, it is impossible to validate the page that has been edited. When the ontology is being edited the whole knowledge base has to be checked for consistency.

If any inconsistencies arise from editing the ontology, they will be reported to the user. This way it is possible to edit the ontology and evolve it to suit the needs of the wiki community and still be on the lookout for possible inconsistencies.

6. DISCUSSION
The discussion of the four use cases implies that the proposed architecture fulfils all requirements. However, every architecture has its strong and weak points. The proposed architecture doesn’t depend on an external ontology because it stores the ontology in the page storage. Because of this users are able to edit the ontology the same way they edit normal pages, giving them almost the same freedom of expression as a normal wiki.

This brings us to the drawback of this kind of architecture, the constrained user input. Because of the validation step the user input is constrained to consistent information. It is still possible to save an inconsistent wiki page, but this will most likely be discouraged, if not prohibited, by the wiki operators. Although the users are able to edit the ontology itself, this extra step forces the user to rethink whether they really want to edit the ontology, or whether they want to re-edit the page.

Another argument in favour of this architecture is its ease of implementation. Because the validator is a separate component that is invoked within the logic it is easy to implement this architecture extension in an existing wiki implementation. Of course, if the wiki implementation doesn’t have semantic capabilities the implementation of the extended architecture will be hindered.

7. CONCLUSIONS
Based on the discussion of several use cases and the analysis of the strong and weak points inherent to the extended architecture we draw the following conclusion: The extended architecture is a simple and adequate architecture for the validation of user input in a semantic wiki. The user input is not constrained to the point where the user won’t edit the wiki pages, and allows much of the freedom enjoyed in the original wiki architecture.

This architecture is only as strong as its reasoner. Therefore it is of importance that an ontology representation and a fitting reasoner can be found. The analysis of ontologies and reasoners is a good topic for further research and enables the architecture to be implemented in mainstream wiki implementations.

Further work in this direction also includes the design of the internal workings of the validator, the extensions to the normal wiki user interface to allow useful and concise feedback to be given about the inconsistencies and the link candidates. Further work also includes the implementation of a prototype.

REFERENCES


SHAWN: Structure Helps a Wiki Navigate.

The semantic Web.

OWL Web Ontology Language Reference.

November 2007.

Ontowiki: community-driven ontology engineering and ontology usage based on wikis.

Agents and the semantic web.

[IZ06] Angelo Di Iorio and Stefano Zacchiroli.
Constrained wiki: an oxymoron?

[KKT06] Kensaku Kawamoto, Yasuhiro Kitamura, and Yuri Tijerino.
Kawawiki: A semantic wiki based on rdf templates.

Reasoning in semantic wikis.

Resource description framework (rdf) model and syntax specification.
http://www.mediawiki.org/wiki/MediaWiki.

How semantics make better wikis.

Pellet: An owl dl reasoner.

[VKV’06] Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, and Rudi Studer.
Semantic wikipedia.

Studying cooperation and conflict between authors with history flow visualizations.