Knowledge Continuous Integration Process (K-CIP)

Hala Skaf-Molli
LINA – Université de Nantes
2 rue de la Houssinière
BP92208, F-44300 Nantes
Cedex 3, France
hala.skaf@univ-nantes.fr

Emmanuel Desmontils
LINA – Université de Nantes
2 rue de la Houssinière
BP92208, F-44300 Nantes
Cedex 3, France
emmanuel.desmontils@univ-nantes.fr

Emmanuel Nauer
LORIA – Université de Lorraine, BP 239,
F-54506,
Vandœuvre-lès-Nancy,
CEDEX, France
emmanuel.nauer@loria.fr

Gérôme Canals
LORIA – Université de Lorraine, BP 239,
F-54506,
Vandœuvre-lès-Nancy,
CEDEX, France
gerome.canals@loria.fr

Amélie Cordier
Université de Lyon
Université de Lyon 1, LIRIS
UMR5205, F-69622, France
amelie.cordier@liris.cnrs.fr

Marie Lefevre
Université de Lyon
Université de Lyon 1, LIRIS
UMR5205, F-69622, France
marie.lefevre@liris.cnrs.fr

ABSTRACT

Social semantic web creates read/write spaces where users and smart agents collaborate to produce knowledge readable by humans and machines. An important issue concerns the ontology evolution and evaluation in man-machine collaboration. How to perform a change on ontologies in a social semantic space that currently uses these ontologies through requests ? In this paper, we propose to implement a continuous knowledge integration process named K-CIP. We take advantage of man-machine collaboration to transform feedback of people into tests. This paper presents how K-CIP can be deployed to allow fruitful man-machine collaboration in the context of the WikiTaaable system.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.0 [**Information Systems Applications**]: General; D.2.5 [**Testing and Debugging**]: Testing tools, Tracing

General Terms

Theory, Verification

Keywords

Ontology, Continuous Integration process, Semantic Wiki, Knowledge Management

1. INTRODUCTION

Social semantic web [17] opened interesting perspectives for man-machine collaboration [12, 20]. It created read/write spaces where users and smart agents can collaborate to improve and maintain knowledge readable by humans and machines. Humans can write documents assisted with smart agents that gather informations from linked data [6] such as Zemanta[1]. Smart agents can perform complex queries on structured knowledge extracted from social tools as DBPedia [2].

Ontologies are the backbone of semantic web and whatever the semantics is – model theoretic versus linked data [11] – ontologies are not an end *per se*; they are dedicated to a task and contribute to the foundation of reasoning systems. However, there is no unique way for expressing any piece of knowledge: an inference is valid if it follows some basic rules in logic and, as well, if it helps in reaching the goals the application is designed for. From the social semantic web point of view, if ontologies are playing a fundamental role, most of the time, they cannot be modified in the social space, or more precisely, not in the same social space than the one used by people. Extracted Wikipedia ontologies such as DBPedia[2] [2] or YAGO [16] cannot be modified directly by Wikipedians. Indeed, a change in the ontology may induce a great change in the application results. This problem is clearly observable in semantic wikis [18] where a user can change a type definition and break all queries or inferences that use this definition. In that way, evaluation of an ontology overpasses the completeness and soundness question and raises the fundamental problem of ontology evolution and evaluation in man-machine collaboration.

In this paper, we propose to implement a continuous knowledge building process named K-CIP. Community control processes such as wiki or social networks are quite convenient for updating or enriching the ontologies. Each user may contribute to information, knowledge construction and regulation. Users are guided by the results of the applications at a given time to make changes which allow new inferences or block some others in the next version. We take advantage of man-machine collaboration to transform feedback of users into tests. These tests allow to evaluate ontology changes in term of user satisfaction. The global change process is obtained by running synchronization protocol on a network of knowledge bases. One node of the network represents the blessed knowledge base where users

WWW 2012 Companion, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1230-1/12/04.

[1]http://www.zemanta.com/
[2]http://dbpedia.org/

are using the system and where interactions are collected as tests. Next, ontology changes can be performed on an isolated node of the network and then propagated to any node of the network. In order to setup continuous integration, a node represents the continuous integration server where tests are gathered and run against ontology changes. If test results are acceptable then ontology changes are propagated back to blessed knowledge base. If failed, proposed ontology changes are cancelled.

This paper presents how K-CIP can be deployed to allow fruitful man-machine collaboration in the context of the WikiTaaable system [4]. It is organized as follows. Section 2 presents related works. Section 3 shows a motivating example. Section 4 gives an overview of the tests system. Section 5 details the K-CIP process. The last section concludes the paper and points future works.

2. STATE OF THE ART

The engineering of ontologies is a central concern in knowledge engineering. Many tools have been proposed to improve ontologies development. Most of these tools (as Protégé [1]) are designed for centralized usage without social collaboration process. However, some collaborative approaches have been proposed.

Co$_4$ [9] defines sharing protocols based on peer-reviewing for finding consensual knowledge; the result is a hierarchy of knowledge bases (KBs), the uppermost ones containing the most consensual knowledge while the lowermost ones are private KBs of contributing users. Co$_4$ supports only hierarchical collaboration and does not support autonomous participants. In addition, it does not allow to represent any kind of processes like an integration processes for instance. Reaching consensual knowledge is not always an easy issue, especially when autonomous participants can enrich knowledge from different sources.

DILIGENT [19] is a methodology focusing on ontology management in distributed environments. It is based on roles and activities. Domain experts build a first basic ontology, then the users have to extend this ontology. Experts must validate users proposals before integrating them into the ontology. Such a system is a kind of validation process for building formal knowledge. As opposed to the K-CIP process, which evaluates an ontology automatically through its application process, in DILIGENT, domain experts validate the proposal and there are no automatic tests to validate them.

OntoWiki[3] is a tool dedicated to support agile and collaborative knowledge engineering. This tool provides centralized collaborative management of an ontology. Changes are tracked and identified. Finally, a rating system allows the community to validate the resulting ontology.

Beyond ontological engineering tools, we are also interested in Continuous Integration process [10] (CI). This is an agile software engineering process aiming at shortening the integration of software. The principle is to allow the developer to make many small updates (and this is even recommended). Each update is subject to a set of unit tests then this update is integrated into the whole software. Next, one or more phases of integration tests (according to their time cost) are processed to ensure the consistent development of the software. A notification system allows the developer to know the state of integration. In case of failure, he/she is quickly notified so that corrections are made quickly.

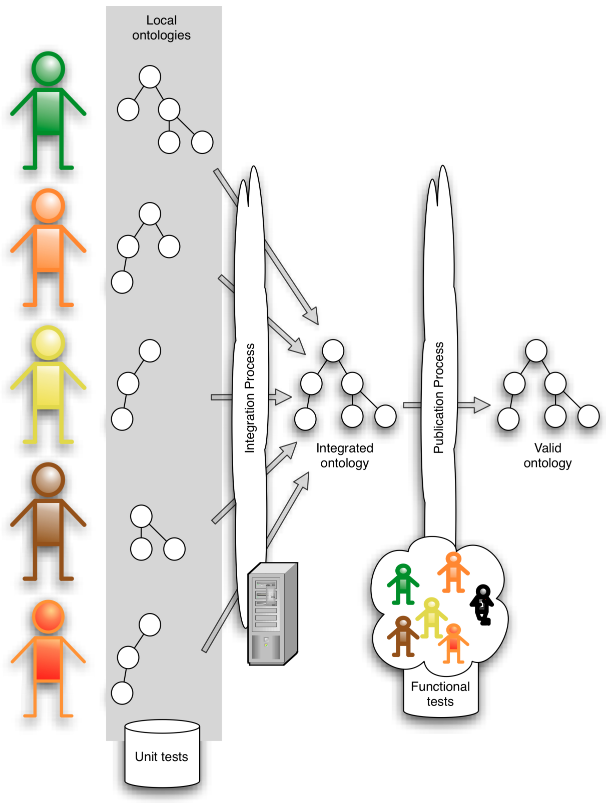


Figure 1: Ontology changes in a continuous integration process

The goal of the K-CIP approach described in this paper is to provide such features for ontology engineering in a social and distributed context. For that, we draw inspiration from continuous integration and agile programming.

3. MOTIVATING SCENARIO AND ILLUSTRATION OF THE APPROACH

In the context of the Kolflow project, our goal is to develop a social semantic space where humans and smart agents can collaborate to build efficient reasoning systems. One way to improve the efficiency of reasoning systems is to enhance the quality of the knowledge they use. Our goal in this paper is to show that we can apply agile software development techniques to ontology engineering. More precisely, we transpose the continuous integration (CI) process to the ontology engineering. This approach allows us to consider the ontology development as an agile distributed collaborative development process as it is shown in Fig. 1.

In this approach, each knowledge engineers (peers) develops a part of the ontology and makes local verifications (with a set of *unit tests*). Every time he/she commits a new set of modifications, the dedicated server automatically launches an integration build to validate the changes and to test the consistency of the ontology. A set of *integration tests* is used to perform the validation. After this step, the server notifies the peer of the result. The peer immediately knows if the changes that he/she committed has been successfully integrated or not. Therefore, checking integration becomes as easy as checking software code. Using an automated integration server does not only makes integration easy, but also guarantees that the ontology is consistent. There is no danger for knowledge engineers to forget to validate their changes. In addition, even if each peer has a partial view of the ontology, the CI process integrates all the views and verifies the consistency of the resulting ontology. Finally, the community of knowledge engineers accepts or not changes according to the functional point of view of the system. Such a phase corresponds to the phase of functional testing in software development.

Integration tests ensure to keep a maximum of "good" properties in the ontology. The main difficulty in this approach is the development of tests (design and build). Therefore, we need ways to gather tests as easily as possible. We argue that tests can be collected using social exchanges. These exchanges can be supported by centralized or distributed architectures. In this paper, we propose to use a distributed architecture, and we use DSMW as a support for this architecture [15, 14].

To illustrate our work, we have taken our examples in the cooking domain. For this, we present the system TAAABLE and the semantic wiki WIKITAAABLE. In the following, we show how, thanks to K-CIP, we can help knowledge engineers to improve the knowledge contained in WIKITAAABLE.

DSMW.

The Distributed Semantic Media Wiki is an extension to Semantic MediaWiki (SMW) [13]. It allows to create a network of SMW servers that share common semantic wiki pages. DSMW manages the propagation and the integration of changes issued on one SMW server to remote servers on the network. The system ensures the consistency of the whole set of replicated pages. DSMW users can create and edit semantically annotated wiki pages as with a regular SMW server. Then they can manage page changes as a software developer does with source code using a distributed version control system. They can work in isolation while editing pages and semantic annotation on a single server. Then they can publish part or all of her own changes by pushing them to DSMW public feeds. They can also subscribe to any remote public DSMW feeds, pull changes from remote servers and integrate them to the local pages. The DSMW extension adds two main features to SMW: an optimistic replication algorithm, and an ontology to manage changes, publication and integration of changes sets.

TAAABLE *and* WIKITAAABLE .

TAAABLE is a web-based application that solve cooking problems. When a user asks for "a dessert with rice and figs" to TAAABLE, the system returns dessert recipes containing rice and figs. If TAAABLE does not have this kind of recipe in his cookbook, it builds one by adapting an existing recipe. For example :

> TAAABLE can retrieve a dessert recipe with rice and mangoes, and recommend the user to replace mangoes by figs to obtain a recipe with rice and figs.

For this, TAAABLE relies not only on its cookbook, but also on a set of adaptation knowledge. The reasoning is performed by a dedicated case-based reasoning engine.

All the knowledge used by TAAABLE is represented in a semantic wiki called WIKITAAABLE [4]. In the current implementation of the system, we use DSMW as the wiki engine for WIKITAAABLE. WIKITAAABLE allows users to visualize the knowledge used in the system and to navigate in the recipe book. The same tool is also used to display recipes and adaptation recommended by TAAABLE.

In the current version of the system, users can directly edit the knowledge *via* the interface of WIKITAAABLE. However, this can be very dangerous because users, even the most experienced, may not necessarily perceive the impact of their changes throughout the system. To illustrate this, consider a simple example :

> We consider that TAAABLE has the recipe for fruit syrup, and is able to adapt the recipe for any fruit. Now suppose that a user decides to classify the tomato as a fruit. Without knowing it, it opens the possibility to create tomato syrup, which is not necessarily desirable.

To avoid such situations to occur, we propose a mechanism to test the impact of a modification of knowledge on the output of a system before permanently integrating the modified knowledge in the base. By doing so, we facilitate the process of maintenance and evolution of knowledge, which improves the overall system quality.

4. TESTS SYSTEM

Our K-CIP approach relies on tests that can be applied to the system. Testing the system consists in running queries and evaluating their results each time a modification of the system is introduced. Tests are expressed as assertions about the result of a query. Test assertions are typically written by the developers of the system and users that introduce modifications. Test data (i.e. queries and their expected results) are collected through the feedback given by end users of the system and stored in the feedback database.

The architecture for collecting test data from users, and for running tests is given in Fig. 2. This architecture is described in the context of TAAABLE.

(1) Users interacts with TAAABLE. TAAABLE proposes recipes and their possible adaptations. An adaptation is a pair (C,M), where C is a case and M a set of modifications to apply on the case. In TAAABLE, C is a recipe and M is a set of substitutions of ingredients $\{i_m\} \to \{j_n\}$, meaning that the set of ingredients $\{i_m\}$ has to be replaced by the set of ingredients $\{j_n\}$ in this recipe. For each modification proposed by the system, users can evaluate through the TAAABLE interface, if it is relevant or not.

(2) Then, we feed a database with the evaluated adaptations: For TAAABLE, an adaptation is relevant (resp. irrelevant) for a user if the user agrees that the modification of recipe by a set of substitutions works (resp. does not work). So, an evaluated adaptation A is an adaptation associated with a positive or negative evaluation.

(3) a user is asked to evaluate if an adaptation is relevant or not and thus, the set of relevant answers or the set of irrelevant answers associated to a query evolves.

(4) In addition, when users modify the ontology, (5) non-regression test-processes are triggered. This process (6) uses the feedback database which stores, for a query Q, the sets of relevant, irrelevant or non evaluated answers. These sets

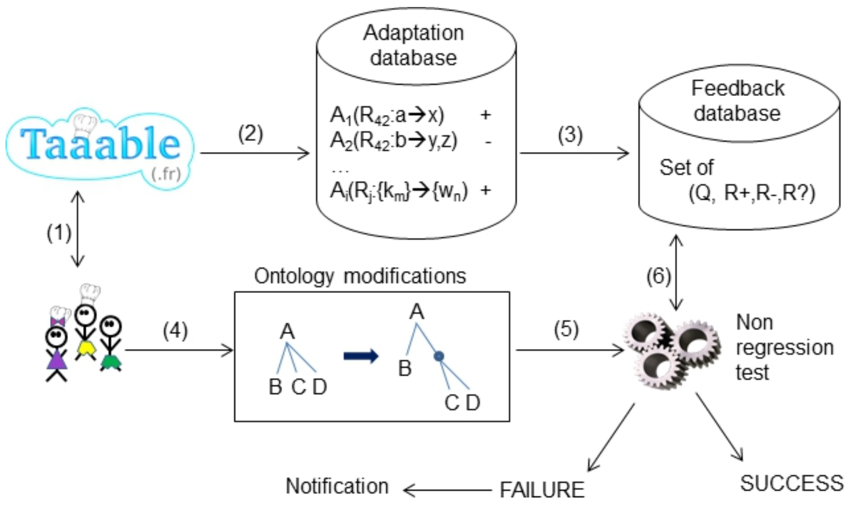


Figure 2: Architecture for collecting and running tests above the Taaable system

are used by "non regression test" processes to evaluate assertions about the state of the system. This enables the system to guarantee that ontology modifications are acceptable (in case of successful test) or not (when some tests fails). In this last case, the modifications are cancelled and a notification is sent to the author. An complementary approach could be to implement some user interactions, as it is for example done in [8], [5].

4.1 Defining tests

A test T is defined as

- a query Q_T;
- a set A_T of assertions $\{A_i\}$ evaluated against the result R_Q produced by the query Q_T when the test is ran.

A test T is said to be successful if all assertions $\{A_i\}$ in A_T are evaluated to true. Assertions in A_T are typically written using three particular data sets that come from the feedback database:

- R^+: the set of answers considered as relevant for Q_T;
- R^-: the set of answers considered as irrelevant for Q_T;
- $R^?$: the set of answers for which no relevant information is available;

Assertions are then defined as logical expressions using set operations on R_Q, R^+, R^-,$R^?$.

4.2 Writing tests

As defined just above, tests are written by defining assertions about the results of a given query.

These tests are written and ran to guarantee that a modification of the ontology does not alter the system behavior. This is done by selecting queries and their expected results from the feedback database and evaluating assertions that compare their effective result to compare with the expected one.

Typical assertions that can be checked are the following:

$Assert(\ R^+ \subseteq\ R_Q\ )$: all the expected answers $R^+$are in the result R_Q, useful to ensure that a modification does not reduce the set of positive answers for a query;

$Assert(\ R^+ =\ R_Q\ )$: the result R_Q is exactly the set of expected answers R^+;

$Assert(\ R_Q\ \cap\ R^- = \emptyset)$: none of the unwanted answers are in the result; useful to ensure that a modification does not introduce unwanted results for a query.

It is also possible to write assertions that compare sets cardinals:

$Assert(|\ R^+ \cap\ R_Q\ | > |\ R^- \cap\ R_Q\ |)$: there is more positive answers than unwanted answers;

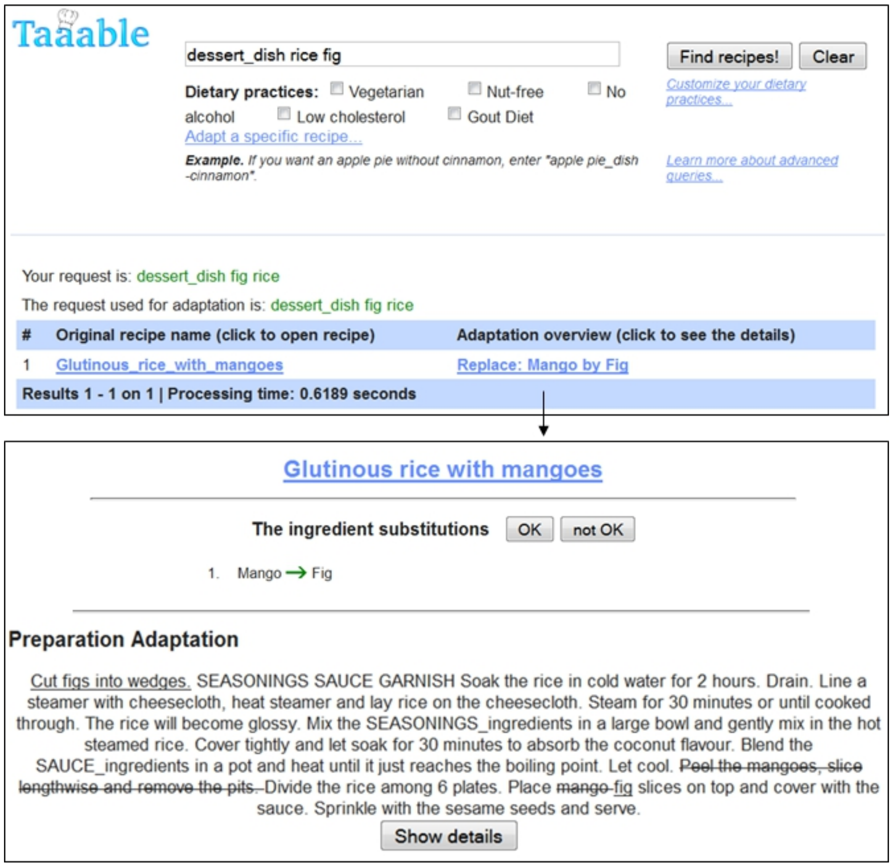
$Assert(|\ R^+ \cap\ R_Q\ | > |\ R^? \cap\ R_Q\ |)$: there is more positive answers than undetermined answers in the result of the query ; useful to check that a modification does not introduce irrelevant noise in the result of a query;

$Assert(\frac{|R_+ \cap R_Q|}{|R_+|} \geq \alpha)$: the query produces at least α% of the expected answers.

4.3 Collecting Test data

In our architecture, we plan to store tests in a wiki. These tests could be written manually by experts and users, e.g. by listing, for a given query, the expected relevant/irrelevant results. However, this option is very time consuming. This is the reason why –and this is an originality of this work– we propose an approach to automatically collect test by gathering user feedback.

In the TAAABLE interface, illustrated in figure 3, we can see that the interface allows users to evaluate a recipe adaptation by saying if the adaptation is "OK" or not. So, when a user evaluates an adaptation as being relevant (by clicking on "OK") the recipe returned and adapted by TAAABLE can be added to $R^+$of the test T (Q, R^+, R^-, $R^?$) where Q is the query which has been used for querying TAAABLE. In the same way, R^-increases when a user evaluates an adaptation as being irrelevant.



dessert_dish rice fig

Dietary practices: ☐ Vegetarian ☐ Nut-free ☐ No alcohol ☐ Low cholesterol ☐ Gout Diet

Customize your dietary practices...

Adapt a specific recipe...

Example. *If you want an apple pie without cinnamon, enter "apple pie_dish -cinnamon".*

Learn more about advanced queries...

Your request is: dessert_dish fig rice

The request used for adaptation is: dessert_dish fig rice

#	Original recipe name (click to open recipe)	Adaptation overview (click to see the details)
1	Glutinous_rice_with_mangoes	Replace: Mango by Fig

Results 1 - 1 on 1 | Processing time: 0.6189 seconds

Glutinous rice with mangoes

The ingredient substitutions OK not OK

1. Mango → Fig

Preparation Adaptation

Cut figs into wedges. SEASONINGS SAUCE GARNISH Soak the rice in cold water for 2 hours. Drain. Line a steamer with cheesecloth, heat steamer and lay rice on the cheesecloth. Steam for 30 minutes or until cooked through. The rice will become glossy. Mix the SEASONINGS_ingredients in a large bowl and gently mix in the hot steamed rice. Cover tightly and let soak for 30 minutes to absorb the coconut flavour. Blend the SAUCE_ingredients in a pot and heat until it just reaches the boiling point. Let cool. ~~Peel the mangoes, slice lengthwise and remove the pits.~~ Divide the rice among 6 plates. Place ~~mango~~ fig slices on top and cover with the sauce. Sprinkle with the sesame seeds and serve.

Show details

Figure 3: The Taaable interface. Queried for a dessert dish, with rice and fig, Taaable proposes to replace mango by fig. After viewing the adapted recipe, the user can give feedback about the substitution (”OK” or ”not OK”).

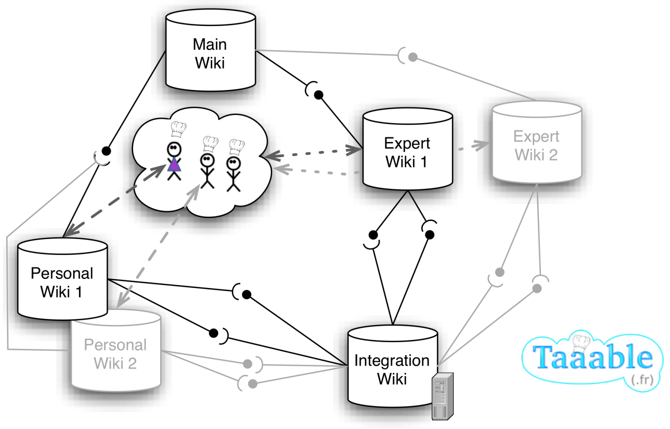


Figure 4: K-CIP overview using DSMW and Wiki-Taable

5. PROCESS SUPPORT

To illustrate the K-CIP process, we use a distributed semantic wiki (DSMW) populated with WikiTaaable data. Therefore, DSMW is used to represent the ontology used by Taaable. DSMW allows the deployment of different processes. Figure 4 presents an architecture deployed in DSMW that allows a knowledge integration process. The process follows these steps:

- Each user has his/her own version of the ontology on his/her own instance of DSMW (”Personal Wiki”) which represent knowledge nodes [7] composing the ontology.
- The different versions of the ontology are combined and tested on an integration server based on an instance of DSMW (”Integration Wiki”). On the Integration Wiki, the test is performed by an intelligent agent. This autonomous agent merge ontologies and run the set of integration tests.
- A set of dedicated wikis (”Expert Wiki”) enables the community to discuss over the proposed solution (the ontology with all personal versions integrated). Each Expert Wiki (over DSMW server) can be seen as a view of the main ontology depending on the topics addressed by each one.

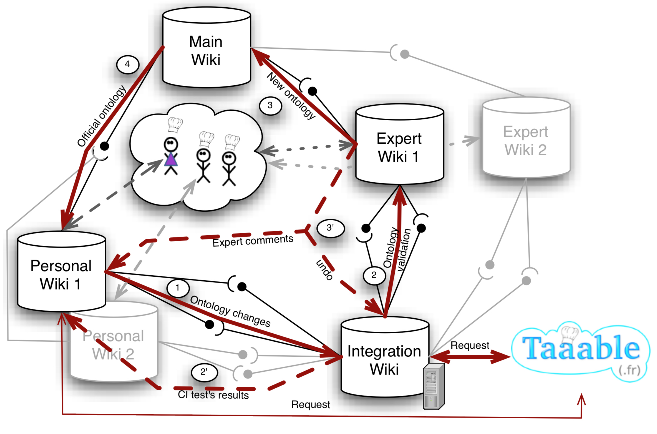


Figure 5: Detailed process of a modification of an ontology

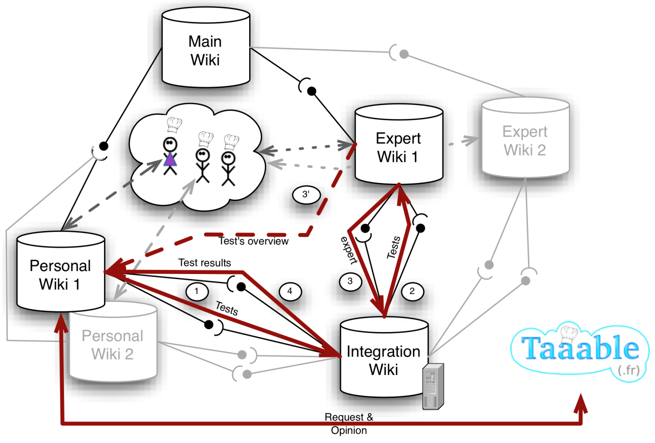


Figure 6: Setting up tests

- Once a consensus is reached (the functional phase is validated), a standard and public version of the ontology is published on a read only public Wiki ("Main Wiki").

In the following, we present a scenario where a user (Mary) modifies an ontology. We use this scenario to illustrate the K-CIP process described above.

Ontology Management.

Mary modifies the ontology of her personal wiki "Personal Wiki 1" (Fig. 5). $\mathcal{O}$ is the initial ontology and $\mathcal{O}'$ is the new ontology obtained by a set of modifications on $\mathcal{O}$. Later, Mary pushes her modifications to the integration Wiki (1) that integrates them in the main ontology and implements the verification campaign with tests. Two cases are possible:

- Tests failure, in this case, the integration wiki informs Mary (2') and explains the reasons for the rejection of its proposal.
- Tests success, in this case the proposed amendments are accepted and $\mathcal{O}'$ is pushed over the expert wiki (2).

In the latter case, the experts group of the community validates $\mathcal{O}'$. If rejected, the author is notified (3'). Otherwise, accepted changes are pushed to the public wiki (3) and users can get it (4). The experts group can only validates or refuses the proposal. No changes are made at this step. Each time a failure happens, a procedure to cancel the edition is enabled.

Tests Management.

Tests are a very important part of K-CIP. Building tests for an ontology follows two ways : the first way is reusing previous tests of $\mathcal{O}$ which are still valid for $\mathcal{O}'$ and the second way is adding new tests specifically for $\mathcal{O}'$.

A successful test for $\mathcal{O}$ can be a successful one for $\mathcal{O}'$. However, depending on the type of test, the properties (like the *sets of responses relevant, irrelevant and unknown*) can evolve. Therefore, a maintenance phase of tests is useful after updating an ontology. Tests are then made available to the community through the Expert Wikis, in order to validate tests updates (like adjusting the different sets). Once validated, changes are updated in the Integration Wiki. Some tests can be finally rejected by the community, the authors are then informed.

Second, any contributor can create a new test (Fig. 6). He/she therefore develops the test by proposing a query, specifying the characteristics of the test and building the sets of the accepted and rejected answers. Once in a satisfactory condition, he/she pushes his/her test to the integration wiki (1). Tests are validated by the expert community (as in the previous case) (2) and integrated in the verification wiki (3) or not. In case of rejection, explanations are suggested to the author of the test (3' - 4).

6. CONCLUSION AND FUTURE WORK

Nowadays, ontology evaluation remains a major challenge for knowledge-based systems. We argue that a collaborative environment implementing a K-CIP is a real advance to go beyond completeness and soundness checking or any formal approach which only consider the ontology.

This paper describes the K-CIP process, a continuous process for enriching and updating ontologies dedicated to a specified task. Impacts of ontology changes on the task are evaluated by a set of tests. In this particular case, the TAAABLE project, a case-based reasoning system on cooking, is used for experiment. This work is conducted within the Kolflow Project which aims at building a social semantic space where humans and intelligent agents can collaborate to achieve various tasks.

Formalization, implementation and experiments have still to be completed.

In addition, several situations remain to be explored. For example, when a modification is rejected by a user or by a community of users, nothing is done yet. Taking into account situations of rejection by users is an important perspective to this work. We are particularly interested in two situations that can cause a bad feedback from users.

The first situation is the situation where different users disagree on a situation, but where everyone wants to keep his point of view. In such a situation, it becomes necessary to manage in parallel several versions of a resources, while ensuring that the overall system will continue to work properly. A new challenges is then to implement processes capable of reasoning in the presence of inconsistent knowledge.

The second situation on which we focus is the "regression" situation. What happens if a change undermines a result that was obtained in the past? Should we consider that the context has changed and that the result obtained in the past is no longer valid? Or should we consider that the proposed modification entails a system regression and therefore, that it should not be applied? To address this problem, we plan to implement a mechanism inspired by the non-regression tests, combined with a strategy of interaction with users to enable them to drive the tests.

These situations are just two examples of the many problems raised by the implementation of collaborative social space that we intend to explore in the project Kolflow. Moreover, K-CIP opens new challenges we are currently studying.

One of these new challenges is to help the user in better understanding how to enrich the ontology in order to make the overall system progresses. As the final task acts as a black box, there could be discord between user intuition for updating the ontology and the evolution of the task during the tests. Implementing a good tracking systems could lead to the definition of a recommendation system to help the user.

Another challenge is suggested by [11] who highlights that semantic web swings between formal model theoretic semantics and a "light" semantics suggested by open-linked data approaches. Completeness and soundness constraints on the ontology could be released to prior task efficiency, enabling, for example, computation of competing solutions which correspond to different interpretation of the ontology.

Thus, social semantic spaces where humans and intelligent agents can collaborate to achieve various tasks is very promising for ontology-based system development, making them more flexible and avoiding ontology obsolescence.

7. ACKNOWLEDGMENTS

This work is supported by the French National Research agency (ANR) through the KolFlow project (code: ANR-10-CONTINT-025), part of the CONTINT research program.

8. ADDITIONAL AUTHORS

Pascal Molli,
LINA Université de Nantes
2 rue de la Houssinière, BP92208,
F-44300 Nantes Cedex 3,
France
pascal.molli@univ-nantes.fr
and
Yannick Toussaint,
INRIA – LORIA,
BP 239, F-54506 Vandœuvre-lès-Nancy, CEDEX,
France
yannick.toussaint@loria.fr

9. REFERENCES

[1] The Protégé Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu/, 2003.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. *The Semantic Web*, 4825:722–735, Jan 2007.

[3] S. Auer, S. Dietzold, J. Lehmann, and T. Riechert. OntoWiki: A tool for social, semantic collaboration. In N. F. Noy, H. Alani, G. Stumme, P. Mika, Y. Sure, and D. Vrandecic, editors, *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at the 16th International World Wide Web Conference (WWW2007) Banff, Canada, May 8, 2007*, volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[4] F. Badra, J. Cojan, A. Cordier, J. Lieber, T. Meilender, A. Mille, P. Molli, E. Nauer, A. Napoli, H. Skaf-Molli, and Y. Toussaint. Knowledge acquisition and discovery for the textual case-based cooking system WIKITAAABLE. In S. J. Delany, editor, *8th International Conference on Case-Based Reasoning - ICCBR 2009, Workshop Proceedings*, pages 249–258, Seattle, United States, July 2009.

[5] F. Badra, A. Cordier, and J. Lieber. Opportunistic Adaptation Knowledge Discovery. In Springer, editor, *8th International Conference on Case-Based Reasoning (ICCBR 2009)*, pages 60–74, July 2009.

[6] T. Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[7] M. Bonifacio, P. Bouquet, and R. Cuel. Knowledge nodes: the building blocks of a distributed approach to knowledge management. *Journal of Universal Computer Science*, 8(6):652–661, Jan 2002.

[8] A. Cordier. *Interactive and Opportunistic Knowledge Acquisition in Case-Based Reasoning*. Thèse de doctorat en informatique, Université Lyon 1, Nov. 2008.

[9] J. Euzenat. Corporate memory through cooperative creation of knowledge bases and hyper-documents. In *Proc. 10th workshop on knowledge acquisition (KAW), Banff (CA)*, pages (36)1–18, 1996.

[10] M. Fowler and M. Foemmel. Continuous integration, http://martinfowler.com/articles/continuousintegration.html, 2005.

[11] P. Hitzler and F. van Harmelen. A reasonable semantic web. *Semantic Web*, 1(1):39–44, 2010.

[12] R. Hoffmann, S. Amershi, K. Patel, F. Wu, J. Fogarty, and D. S. Weld. Amplifying community content creation with mixed initiative information extraction. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 1849–1858, New York, NY, USA, 2009. ACM.

[13] M. Krötzsch, D. Vrandecic, M. Völkel, H. Haller, and R. Studer. Semantic wikipedia. *Journal of Web Semantics*, 5:251–261, December 2007.

[14] H. Skaf-Molli, G. Canals, and P. Molli. DSMW: a distributed infrastructure for the cooperative edition of semantic wiki documents. In A. Antonacopoulos, M. Gormish, and R. Ingold, editors, *ACM Symposium on Document Engineering (DocEng 2010)*, pages 185–186, Manchester, Royaume-Uni, 2010. ACM.

[15] H. Skaf-Molli, G. Canals, and P. Molli. DSMW: Distributed Semantic MediaWiki. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije,

H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *7th Extended Semantic Web Conference (ESCW 2010)*, volume 6089 of *Lecture Notes in Computer Science*, Heraklion, Grèce, 2010. Springer.

[16] F. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[17] Tom and Gruber. Collective knowledge systems: Where the social web meets the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):4 – 13, 2008. Semantic Web and Web 2.0.

[18] M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 585–594, New York, NY, USA, 2006. ACM.

[19] D. Vrandecic, S. Pinto, C. Tempich, and Y. Sure. The diligent knowledge processes. *Journal of Knowledge Management*, 9(5):85–96, 2005.

[20] F. Wu and D. S. Weld. Autonomously semantifying wikipedia. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 41–50, New York, NY, USA, 2007. ACM.