Ontology-Driven Mindmapping

Petr Křemen
Czech Technical University in Prague, Faculty of Electrical Engineering
Technická 2, 16627 Prague, Czech Republic
petr.kremen@fel.cvut.cz

Pavel Mička
Czech Technical University in Prague, Faculty of Electrical Engineering
Technická 2, 16627 Prague, Czech Republic
mickapa1@fel.cvut.cz

Marek Šmíd
Czech Technical University in Prague, Faculty of Electrical Engineering
Technická 2, 16627 Prague, Czech Republic
smidmare@fel.cvut.cz

Miroslav Blaško
Czech Technical University in Prague, Faculty of Electrical Engineering
Technická 2, 16627 Prague, Czech Republic
blaskmir@fel.cvut.cz

ABSTRACT

Mindmaps have been a popular means for intuitive and user-friendly conceptual modeling for several years. This paper proposes a mindmapping tool driven by semantic web ontologies that allows to assign meanings to the mindmap nodes and, even more importantly, guide the user through mindmap creation by proposing ontological concepts and roles relevant to each mindmap node. The tool is backed by OWL ontologies that allow to capture domain knowledge semantics precisely and rigorously. The tool is being developed as a part of the MONDIS project aimed at efficient knowledge management of immovable cultural heritage failures records. As such, the tool is currently used by civil engineering experts for collaborative authoring of structural damage records.

Categories and Subject Descriptors

H.3.4 [**Information networks**]: Semantic Web, ontologies; D.2.3 [**Standards**]: OWL

General Terms

Design, Human Factors

Keywords

semantic web, OWL, ontologies, mindmaps, linked data

I-SEMANTICS 2012, 7th Int. Conf. on Semantic Systems, Sept. 5-7, 2012, Graz, Austria

1. INTRODUCTION

Mindmapping is popular – mindmaps [2] are easy to use and graphically appealing way to capture ideas and knowledge. They are also suitable for collaboration, as observed in [1]. Their success is documented by the amount of mindmapping tools that have appeared during past years, both commercial (like MindManager[1], or XMind Pro[2]) and open source (like FreeMind[3], WiseMapping[4], or XMind[5]).

A significant drawback of mindmaps, also addressed in [7], is the lack of formal semantics which leads to ambiguous interpretations and thus reduces their reusability. Also, the unconstrained nature of mindmaps prevents the mindmapping tools to guide the author through the mindmap creation process in cases when the captured knowledge is supposed to comply with some structure.

We faced these issues at the beginning of the MONDIS project[6], that aims at developing a knowledge based system for recording and analyzing structural failures of immovable cultural heritage objects. During the first months of the project, the domain knowledge model was developed in the form of domain OWL ontologies [5] capturing the semantics of cultural heritage objects, their construction characteristics, associated failures, interventions that repair/prevent these failures, and other phenomena, for example[7]:

Tower **subClassOf** Component (1)

hasMaterial **inverseOf** isMaterialOf (2)

owl:Thing **subClassOf** (hasMaterial **only** Material) (3)

(hasMaterial **some** Material) **subClassOf** Component (4)

[1] `http://www.mindjet.com/products/mindmanager`, cit. 10.4.2012
[2] `http://www.xmind.net/pro`, cit. 10.4.2012
[3] `http://freemind.sourceforge.net`, cit. 10.4.2012
[4] `http://www.wisemapping.org`, cit. 10.4.2012
[5] `http://www.xmind.net`, cit. 10.4.2012
[6] `http://www.mondis.cz`, cit. 15.4.2012
[7] OWL fragments in this paper will be presented in the variant of Manchester syntax used in Protégé ontology editor, available at `http://protege.stanford.edu`.

To validate the domain model, domain experts were supposed to develop example records of structural failures that comply with the model, for this purpose also transformed into mindmaps. An example of such mindmap is in Figure 1. The figure describes the structural failure of the Pisa tower and its components.

To author these examples, civil engineering experts were using Freemind, an easy to use and intuitive mindmapping tool. Unfortunately, the mindmap authoring process turned out to be rather slow, as domain experts had to look up suitable terms and their dependencies in the knowledge model. Moreover, they often overlooked an existing term and its definition in the knowledge model and created a new term with the same meaning, in which case terms in different examples had to be aligned and unified later on. Another delay was caused by subsequent transformation of mindmap example records into ontological descriptions and checking their consistency w.r.t. the model to provide feedback to the example authors. Due to these obstacles, only a few example damage records were developed – their amount was insufficient for reliable validation of the ontology quality.

Having these pitfalls in mind, this paper proposes a mindmapping tool driven by ontologies. In our scenario, the tool speeds up damage example creation, which in turn speeds up ontology validation. Section 2 presents our proposal for collaborative semantic mindmapping. A prototypical tool that partially realizes the architecture, called OntoMind, is presented in Section 3. The usage of the tool is demonstrated in Section 4 on example mindmaps of structural failure records and concluding remarks showing our future directions are presented in Section 5.

2. PROPOSED ARCHITECTURE

Our proposal for the ontology-driven mindmapping system – OntoMind – stems from the following premises:

Intuitive Visualization. The system should use intuitive knowledge visualization. We propose to use mindmaps as a suitable knowledge authoring/visualization instrument – they are generally accepted and understandable.

Guidance. The system should guide users through knowledge authoring process using formal knowledge. Vast amount of shared formal knowledge is captured by semantic web ontologies [6]. Semantics of ontologies are typically expressed by W3C recommendations RDF(S) and OWL, thus allowing intelligent processing using various inference engines. We propose to use domain specific semantic web ontologies to drive mindmap authoring. This involves both guiding the user by suggesting ontological inferences when creating a new node in a mindmap, as well validating domain specific integrity constraints (as proposed in [8] for OWL ontologies).

Online Collaboration. The system should be web-based and it should allow online collaboration and publishing of shared knowledge. During the last five years, Linked Data technologies [3] have emerged and reached significant amount of data by now[8]. We propose that the system should make use of existing Linked Data while guiding users through mindmap authoring (see Guidance above). Similarly, it should be ready to easily publish mindmaps as Linked Data upon users requests.

Offline Operation. Although the system is meant as an online web application, we also expect offline scenario, i.e. users not connected to the internet should be able to create their mindmaps offline, store them into a local client-side storage and the system should synchronize local mindmap changes with the OntoMind server upon reconnecting. Ontological reasoning will be disabled or limited in this setup.

The proposed architecture of the system is shown in Figure 2. The system is split into a client part and a server part. For the sake of quick response times, as well as to support offline operation, the client part involves the full mindmap editor visualization and editing component that uses two APIs to operate with the server:

Mindmapping API serves for management of mindmap lifecycle, i.e. creating, persisting, removing of mindmaps, as well as transforming a mindmap into an ontology, checking its consistency and integrity constraint validations.

Ontology API evaluates ontology queries to infer possible suggestions during editing of a new node. This API does not persist any mindmap data; instead, it stores the data only in a non-persistent session.

Both APIs make use of an ontology layer that provides basic ontology services (like debugging, SPARQL queries, etc., as discussed in [9]) on top of a triple store (e.g. SDB[9], or Sesame[10]). A mindmap author can indicate that a mindmap should become public, in which case it becomes available through the linked data interface to the RDF triple store.

2.1 Collaboration and Knowledge Approval

Let's take a closer look at the online collaboration support, as an important community instrument to produce shared high-quality knowledge. The data model that defines basic relationships between users, mindmaps and knowledge is depicted in Figure 3.

Each mindmap is associated with its *reasoning scope* that represents all ontologies (including integrity constraints) w.r.t which the reasoning will be performed. A reasoning scope can be shared by many mindmaps of the same type, e.g. in the MONDIS project, the reasoning scope will consists of the component ontology, material ontology, failure ontology and the core data model. Each mindmap has associated an owner and possibly several collaborators – in each case, a mindmap is edited by at most one collaborator at a time. The mindmap owner decides on publishing the map through the linked data endpoint.

In addition to external ontologies, a reasoning scope can contain two forms of ontological knowledge created by OntoMindusers – a shared ontology and a sandbox ontology. First of all, users have to be given freedom in defining their own terminology. For this purpose, each user develops his/her own sandbox ontology, hidden to the other users, for each

[8]295 Linked Data datasets contained more than 31 billions RDF triples, as of September 2011, as presented in `http://www4.wiwiss.fu-berlin.de/lodcloud/state`.

[9]`http://incubator.apache.org/jena`, cit. 22.4.2012

[10]`http://www.openrdf.org`, cit. 22.4.2012

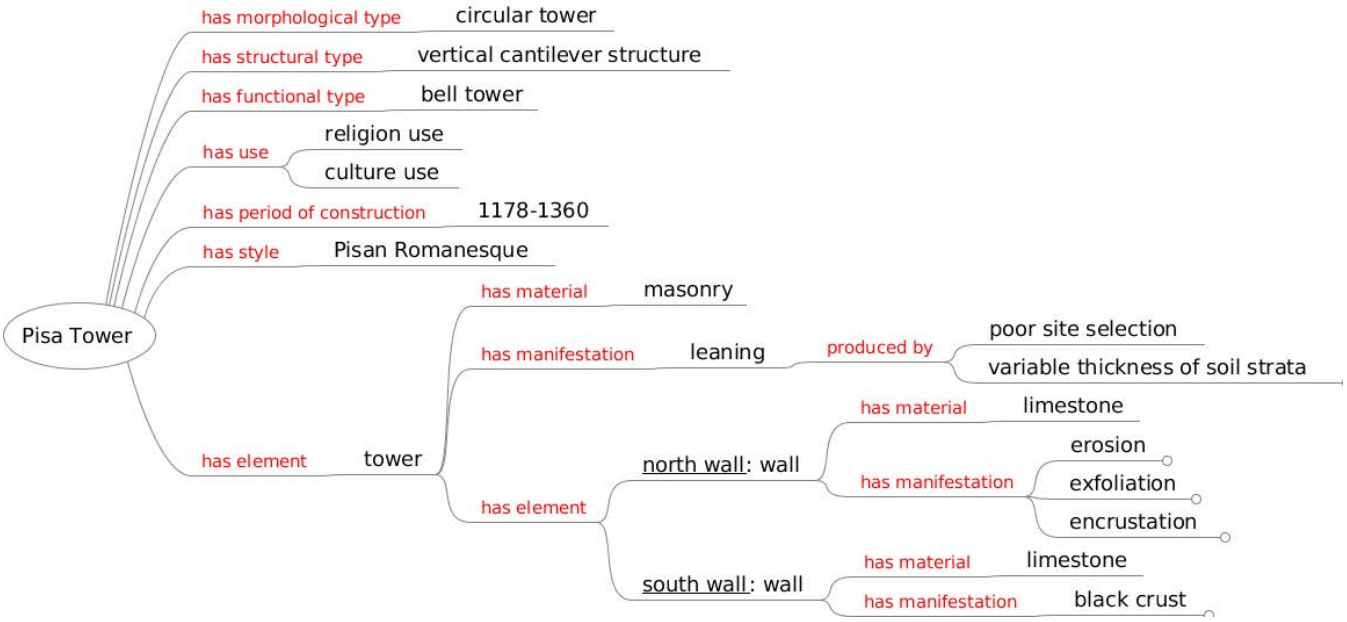

Figure 1: The Pisa tower example.

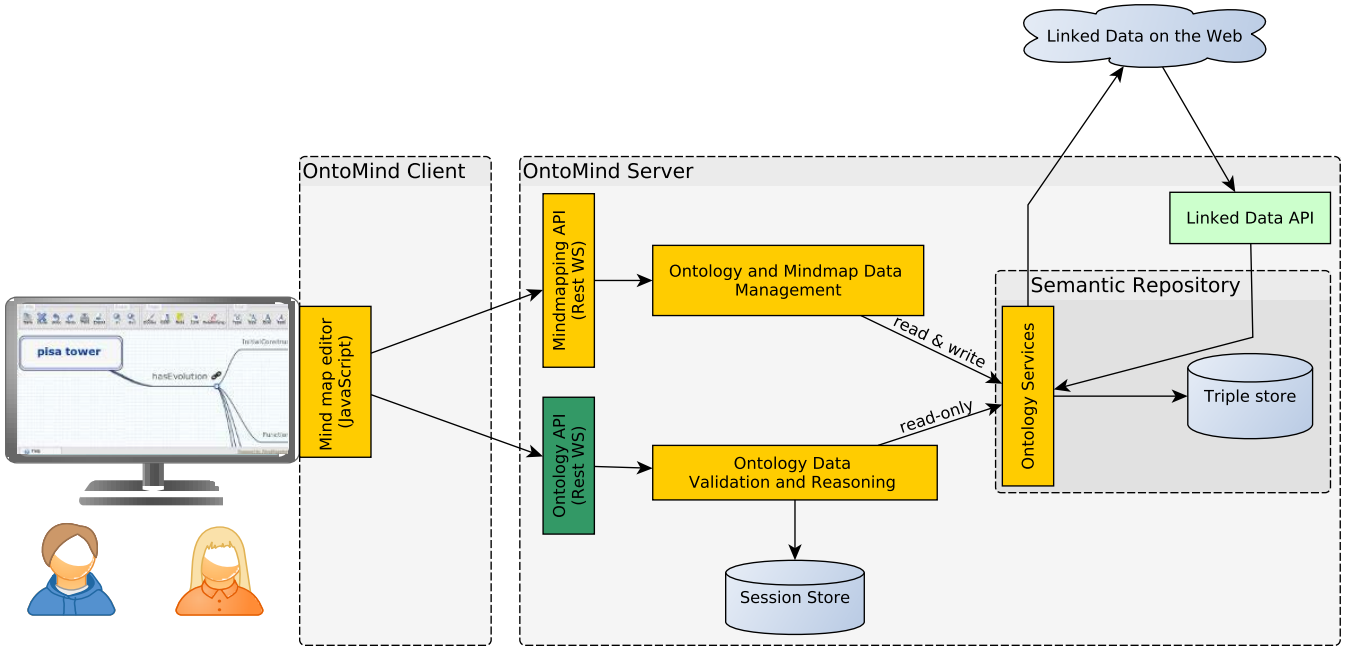

Figure 2: Semantic Mindmapping Architecture. Nodes denote components and links correspond to method calls.

reasoning scope. The user's sandbox ontology is used for reasoning and integrity constraint validation during editing a mindmap belonging to the reasoning scope. From time to time, community of users maintaining the current reasoning scope might decide to revise individual sandbox ontologies and approve their parts as generally valid. As a result of the approval, these sandbox parts, involving cleaned, validated and semantically disambiguated terminology, are included into the shared ontology. Whenever a user is editing a mindmap, the shared ontology belonging to the associated reasoning scope is used for reasoning and integrity constraint checking.

2.2 Using Mindmaps for Ontologies

2.2.1 Data Mindmaps

As we designed the data mindmap – ontology correspondence, each node of a data mindmap in OntoMind corresponds either to an individual in the ontology, or to an object role, which links two individuals together. This represents

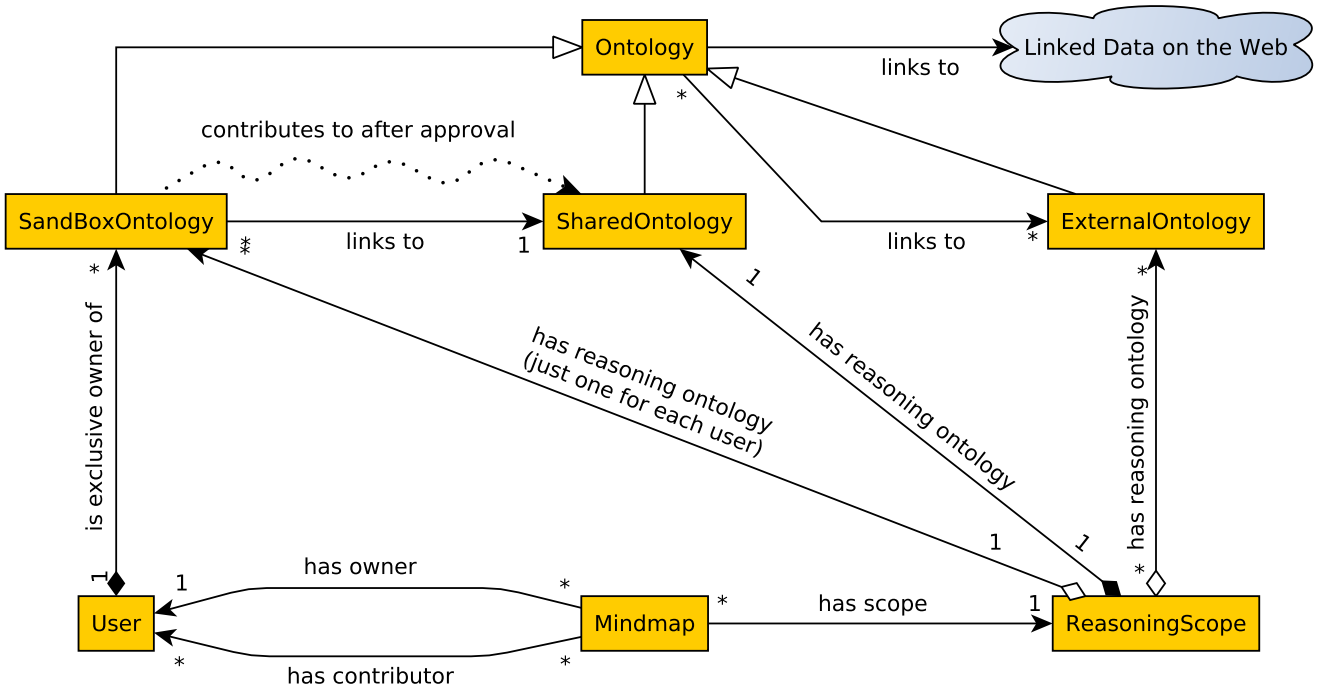

Figure 3: Data Model for Semantic Mindmapping. Nodes denote data classes and links denote relationships/dependencies.

semantic contents of the mindmap. Nodes representing individuals have to be alternated with the ones representing object properties. Besides the nodes' correspondence to ontological terms, they are labeled with text, which does not have any semantic interpretation, but helps identifying the nodes to the user.

Next to the nodes representing individuals, OntoMind can show the hierarchy of ontology classes, which allows assigning a class to the individual (making it an instance of that class). Similarly, next to a node representing an object property a tree of object roles can be shown, allowing it to select one. For details of the tree selection component, see Section 3.1.

Having the mindmap only as a tree structure would disallow reuse of individuals, hence there is the option of creating a graphical link. This allows connecting an object role node to a node of an individual somewhere else in the mindmap.

Thus the ontology, which can be built with such a data mindmap, consists of individuals, possibly being instances of selected classes, linked together by object roles. The text labels of the nodes serve as label annotations of the individuals and object properties.

For an example of data mindmap, see Figure 1.

2.2.2 *Terminological Mindmaps*

Mindmaps of another type serve for editing terminologies, i.e. the domain knowledge. These terminological mindmaps represent either the hierarchy of ontology classes or the hierarchy of object properties[11].

The central node always represents the top-level element in the hierarchy, being either the owl:Thing for classes, or the owl:topObjectProperty for object properties, or any subclass/subproperty of these.

Then the hierarchy is simply represented by the mindmap tree, where links (in the direction from the root) represent "is a superclass of" or "is a superproperty of", respectively. In case a node has more parents than one, additional parents can be attached by graphical links.

3. PROTOTYPE

We have developed a prototype mindmapping tool that reflects some of the needs mentioned in Section 2. The tool guides users through the mindmap authoring process using a whispering component introduced later in this section. Also, the tool immediately synchronizes the content of the mindmap with the underlying OWL ontology.

3.1 Tree Autocomplete Component

One of the most important features of OntoMind from the user interaction point of view is the autocomplete tree component, which guides an user through the mindmap elaboration process by suggesting him all viable terms that might refine the current knowledge.

3.1.1 *Tree Structure*

Traditional autocomplete components, well known from search engines, consist of two main parts — an input field and a list of alternatives filtered by the user input. While

[11]While expressive ontological constructs (e.g. property transitivity (inverse)functionality, (a)symmetry, property chains, class disjointness/complement or existential/universal/cardinality restrictions) are hard to visualize and thus hard to grasp by non-experts in ontological engineering, class/property taxonomies proved to be easily understandable by them in case of MONDIS.

creating an ontology driven mindmap, this approach is not sufficient, because the terminological knowledge (e.g. class or object property taxonomy) in most applications forms a direct acyclic graph (DAG). Hence the suggestions in our autocomplete component are represented as a tree (with possibly duplicate subbranches, where a node has more than one parent in the respective DAG).

The tree is not only straightforward for the visualization of the suggestions hierarchy, it also allows a convenient way to filter the items in respect to the user input — always the whole path from the root to the matching term is displayed to allow user to select the desired item, as well as more generic alternatives. Also all descendants of the matching term are displayed providing user with more specific knowledge (the system provides user with description of every term using a DBpedia endpoint).

The user might also decide to refine the current terminology hierarchy by adding a new term, or even not to use the hierarchy at all. If he decides for the latter alternative, than the system will be able to provide only a limited completion functionality.

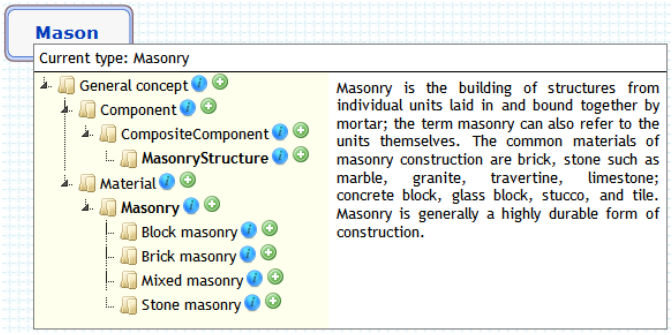

Figure 4: Tree tooltip filtering. Clicking on the triangle left to an OWL class name shows its direct subclasses in the tree.

3.1.2 *Ontological Identity*

The system operates with an ontological identity of the node, which is gained by selecting an appropriate term from the autocomplete tree and is used by the reasoning and suggestion engine of the editor. The term selection binds the mindmap node to a given object property, or creates a new individual, if the term represents an OWL class. On the contrary, the label of a node only improves readability of the map and is not used by the system to infer suggestions.

3.2 Implementation

The client side of OntoMind is implemented using widely supported browser technologies – HTML5[12], SVG[13], and JavaScript. The server side is implemented in Java and built using Maven build manager. The project sources can be downloaded at `http://krizik.felk.cvut.cz/km/ontomind`.

We suppose that OntoMind will be used in many different systems and environments. The current implementation can be deployed out-of-the-box as a standalone application or as a Liferay Portal Portlet. Challenges in maintaining both variants are described in Appendix A.

[12] `http://caniuse.com/#cats=HTML5`, cit. 10.4.2012
[13] `http://caniuse.com/#cats=SVG`, cit. 10.4.2012

3.2.1 *WiseMapping*

OntoMind is based on WiseMapping, a free open-source web based mindmapping tool. WiseMapping can be obtained at `http://www.wisemapping.org/` and tried out at `http://www.wisemapping.com/`. Using Spring framework[14] and Hibernate persistence[15] as a backend, it provides a user with a web frontend generated via Spring Web MVC[16] and JavaServer Pages[17] (JSP), utilizing JavaScript and SVG[18] as a drawing API (being a part of the upcoming HTML5[19] standard).

WiseMapping is a general mindmapping tool providing comfortable user interaction using keyboard shortcuts and attractive graphical output. It features various options for node appearance (font, color, icon, shape, attaching notes and links, etc.). In addition to the basic tree structure of a mindmap, a relationship between two arbitrary nodes can be established, connecting them with a dotted line. A node can be moved around the mindmap with a mouse to reattach it somewhere else. However, this feature has been temporarily disabled in OntoMind since it could break consistency from semantic point of view. The positions of nodes are layed out automatically, respecting the tree structure.

3.2.2 *Portlet Deployment*

One way how to deploy OntoMind is as a Portlet[20] (a standard for pluggable user interface web components that are managed and displayed in a web portal), which fits MONDIS project well, since its website is based on the Liferay Portal[21]. Currently, OntoMind is running fully functional on the internal pages of MONDIS project's portal.

OntoMindPortlet deployment has several advantages over its standalone deployment. The most important of them are:

- Easy portability and deployment – as a Portlet, OntoMind can be easily installed on any server running a portal.
- User management – the user credentials can be reused by many applications within the portal; the portal provides authentication and authorization based on complex grouping and collects user information.
- Integration with other Portlets – OntoMind can share the same web page with Portlets providing other services.
- Customization – users can personalize the website by setting e.g. a look and feel or a language.

[14] `http://www.springsource.org/`, cit. 10.4.2012
[15] `http://www.hibernate.org/`, cit. 10.4.2012
[16] `http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html`, cit. 10.4.2012
[17] `http://www.oracle.com/technetwork/java/javaee/jsp/`, cit. 10.4.2012
[18] `http://www.w3.org/TR/SVG/`, cit. 10.4.2012
[19] `http://dev.w3.org/html5/spec/`, cit. 10.4.2012
[20] As defined in JSR 286: Portlet Specification 2.0 at `http://www.jcp.org/en/jsr/detail?id=286`, cit. 10.4.2012
[21] A free and open-source enterprise portal supporting Portlet specification v2.0, developed by Liferay, Inc., which can be found at `http://www.liferay.com/`.

4. USE CASE

One of the goals of the MONDIS project is to develop an OWL ontology for recording and analyzing structural failures of cultural heritage objects. In this section we describe the structure of the currently developed MONDIS ontology and guide the reader through creation of an example record demonstrating how the OntoMind prototype works.

4.1 MONDIS domain ontology

The currently developed MONDIS ontology captures the semantics of cultural heritage objects, their characteristics, associated failures, interventions that repair/prevent these failures, etc. A simple conceptual model representing the most important parts of the ontology is depicted in Figure 6. In the figure, each box represents a top-level class of the ontology, while links represent significant properties that connect instances of the respective classes represented by the boxes.

4.2 The Pisa tower example record

The interaction of MONDIS domain expert with OntoMind can be demonstrated on the structural failure record of the Pisa tower, Freemind version of which is depicted in Figure 1. The failure record describes leaning of the tower caused by poor site selection and variable thickness of the soil strata. The leaning of the tower is also related to the damage (such as exfoliaton and encrustation) of the walls – tower's components.

When OntoMind user triggers the editor action "create new mindmap", a simple mindmap with a single central node is created. In addition, OntoMind provides button to trigger action "show model", which presents to the user the simple conceptual model of the underlying ontology depicted in Figure 6. Currently, it is possible to show a static image of the model, which was previously imported by the OntoMind administrator.

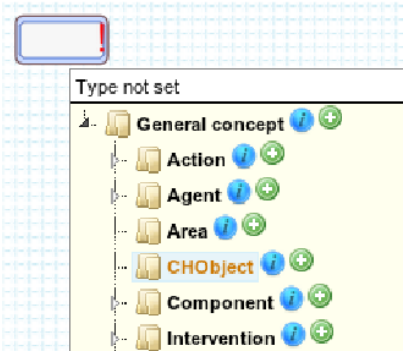

Figure 5: Selection of CHObject term.

To benefit from guidance provided by OntoMind, the user must double-click on the mindmap node and select an ontology class from the tree autocomplete component, as described in Section 3. In MONDIS structural failure examples, CHObject (cultural heritage object) is typically selected as the central node by example authors, as shown in Figure 6. By selecting the class CHObject, a new OWL individual **x** is created, which becomes the ontological identity of the node. Additionally, the class assertion **x**: CHObject is added to the mindmap ontology and the label of the node inherits the name of the selected class, thus "CHObject".

If needed, it is possible to assign a different label to a node, by clicking on the label and renaming it. As a result a new annotation axiom is added to the sandbox ontology, annotating the OWL individual **x** with **rdfs:label** property and value "Pisa Tower".

As shown in Figure 7, we renamed the central node of the mindmap to "Pisa Tower" and assigned an ontology identity to it which is indicated by missing exclamation mark over the node. On the other side, the figure depicts two nodes with labels "has location" and "Italy" which are not connected to an ontological identity, thus they are marked with exclamation marks. MONDIS ontology (see conceptual model in the Figure 6) defines neither object property hasLocation, nor class such as Area being a potential superclass of Italy. Nevertheless, we can define a new object property hasLocation by clicking on the plus sign next to the generalProperty (representing owl:topObjectProperty) node of the tree component as shown in Figure 7. Similarly, a new class can be defined by clicking on the plus sign next to the GeneralConcept (representing owl:Thing) node as shown in Figure 5, e.g. to create the class Area and then to use the plus sign next to the newly created Area node to add the class Italy. These additions result in the following OWL axioms into the user sandbox ontology:

$$\begin{aligned} &\text{hasLocation } \mathbf{subPropertyOf} \text{ owl:topObjectProperty},\\ &\text{Area } \mathbf{subClassOf} \text{ owl:Thing},\\ &\text{Italy } \mathbf{subClassOf} \text{ Area}. \end{aligned}$$

Figure 7 demonstrates another important feature of OntoMind. When selecting an ontology identity for the node connected to another node representing OWL individual (in this case individual belonging to class Tower), the tree component offers taxonomy of object properties instead of taxonomy of classes. Moreover, the tree component offers only relevant object properties for the OWL individual (see Component node of our conceptual model from the Figure 6) . For example hasMaterial object property is included in the selection based on inference from mindmap ontology axioms (1) – (4) shown in Section 1.

Similarly, Figure 8 demonstrates selection of a class based on ontological identity of the node representing the object property hasMaterial. Using the mindmap ontology axiom (3) from the Section 1 it is inferred that only subclasses of the class Material are relevant for the selection. In addition, the selection is narrowed down by the search string "Mason". Compare it to the selection with the same search string but node not attached to the ontological context shown in Figure 4.

5. CONCLUSIONS

This paper proposed a software architecture, together with a prototype implementing some of the presented ideas, that allows for easy and efficient collaborative mindmapping that is driven by semantic web ontologies. In our scenario, making failure record authoring process smoother and faster is one of the significant advantages of the introduced ontology-driven approach over the traditional mindmapping.

The next advantage of the ontology-driven approach is its exploratory nature [4]. Without the guidance of the system, users tend to use their own terminologies, which might be heterogeneous or even mutually incompatible or may lead to ambiguities. While guided by the editor, the user is mo-

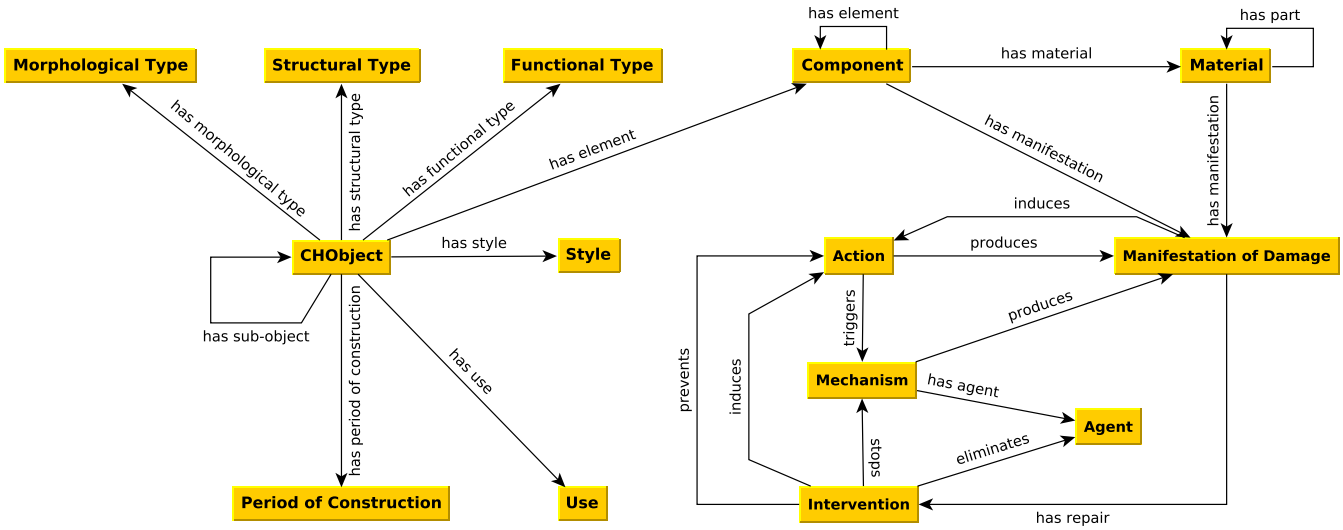

Figure 6: MONDIS conceptual model.

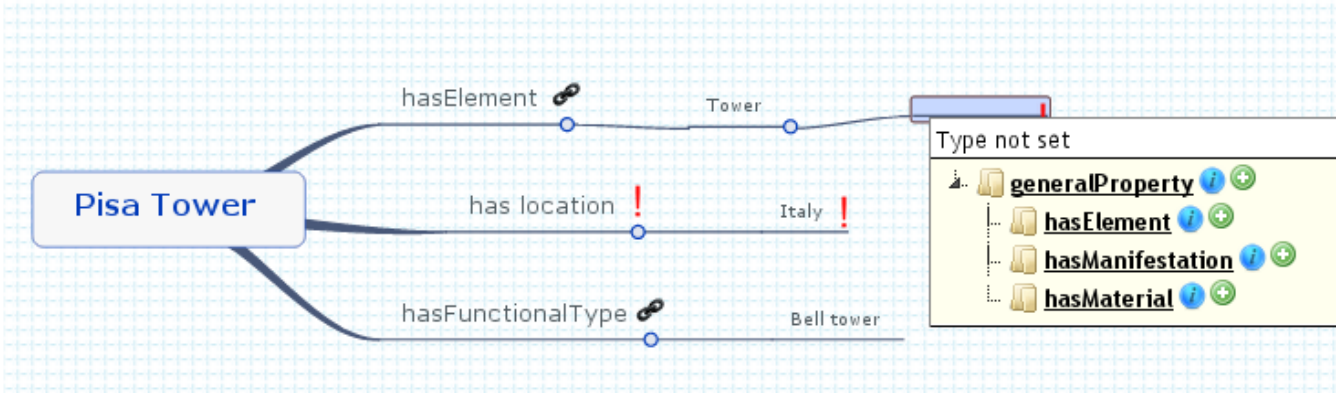

Figure 7: Selection of object property based on the context.

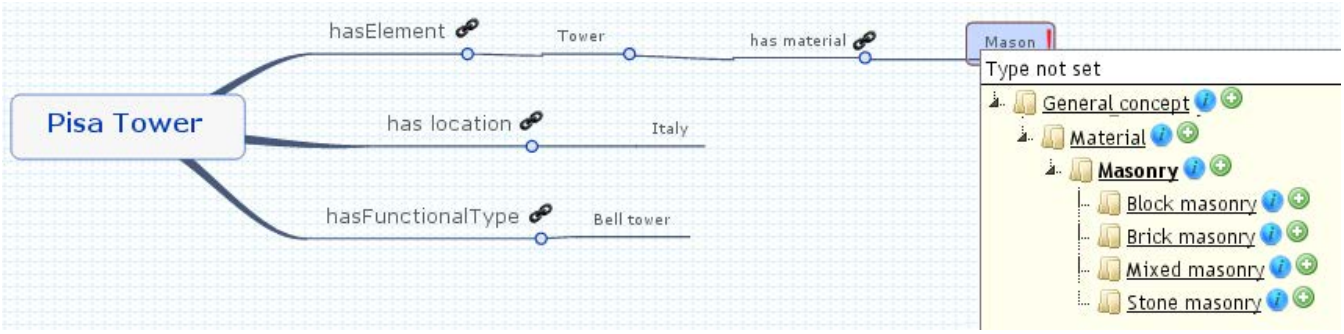

Figure 8: Selection of a class based on the context.

tivated to explore the common knowledge, search in it for appropriate terms (autocomplete field, tree traversal), investigate it and learn it by mindmap authoring. The more complex the mindmap is, the more convenient for the user is to use, extend and refine the current knowledge, rather than create his/her own terminology. Altogether this leads to a homogeneous knowledge controlled by the community.

In the next steps, we will extend the prototype to fully reflect the architecture described in Section 2, providing full OWL consistency checking, integrity constraints checking, workflows for sandbox changes approval and conflict resolution, as well as the possibility to edit terminological mindmaps.

6. ACKNOWLEDGMENTS

This work has been supported by the grant "Defects in immovable cultural heritage objects: a knowledge-based system for analysis, intervention planning and prevention", No. DF11P01OVV002 of the Ministry of Culture of the Czech Republic.

7. REFERENCES

[1] 10 Values of Collabortative Mind Mapping. [online], Sept. 2009. Available at `http://blog.mindjet.com/2009/09/the-true-value-of-collaborative-mind-mapping`, cited 11/4/2012.

[2] T. Buzan and B. Buzan. *The Mind Map Book*. BBC Books, London, 2 edition, 1995.

[3] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, 1 edition, 2011.

[4] G. Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, Apr. 2006.

[5] B. Motik, P. F. Patel-Schneider, and B. Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax [online]. W3C Recommendation, W3C, 2009. Available at `http://www.w3.org/TR/2009/REC-owl2-syntax-20091027`, cit. 11.1.2012.

[6] N. Shadbolt, T. Berners-Lee, and W. Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[7] V. Siochos and C. Papatheodorou. Developing a Formal Model for Mind Maps. In *First Workshop on Digital Information Management*, pages 39–44, Mar. 2011.

[8] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity Constraints in OWL. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[9] T. Tran, P. Haase, H. Lewen, Ó. Muñoz-García, A. Gómez-Pérez, and R. Studer. Lifecycle-support in architectures for ontology-based information systems. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference (ISWC'07/ASWC'07)*, pages 508–522, Berlin, Heidelberg, 2007. Springer-Verlag.

APPENDIX

A. PORTLETIFYING OF WISEMAPPING

Converting OntoMind based on WiseMapping from standalone web application to Portlet required the following four most important steps:

- Configuration – setting up a Portlet deployment descriptor and some other Liferay Portal specific configurations.
- User management – instead of asking user to authenticate and retrieving user information from its own database, it has to rely on authentication, authorization, and user information provided by the portal.
- Web controllers – since JSP's of WiseMapping were using Spring MVC controllers for Java Servlets, the controllers had to be replaced with their Portlet versions (which support Portlet request and response model).
- URL addresses – instead of using absolute URL's for navigating inside the Portlet, we had to use Portlet specific method for generating URL's (JSP tags `<portlet:actionURL>` and `<portlet:renderURL>`).