

# Using Ontologies for Users-Groups Matching in An Annotation System

Danilo Avola, Paolo Bottoni, Amjad Hawash  
Department of Computer Science, Sapienza University of Rome  
Via Salaria 113, 00198, Rome, Italy  
(avola, bottoni, hawash)@di.uniroma1.it

March 6, 2014

## Abstract

Representing a domain knowledge through an ontology allows people and systems to share a common corpus of knowledge for reasoning and communicating. We have recently introduced groups to the MADCOW annotation system, and in this paper we describe how the integration of ontologies allows more refined annotations to be posted on groups focused on some domain. In particular, concepts of the domain ontology can be used as tags, which are integral components of annotations, thus allowing more semantically significant queries for retrieving annotations on specific topics. Services for promoting participation to groups of potentially interested users can also be fostered by the adoption of domain ontologies. For example, authors of annotations systematically using tags referable to some existing ontology can be invited to join groups based on that ontology. Conversely, the analysis of texts of annotations, or of annotated texts, associated with some ontology, can represent a useful addition to construct associations between the ontology concepts and the set of lexical terms associated with them.

**Keywords:** Web annotation, Ontologies, Matching.

## 1 Introduction

Digital annotation is the process of adding information to contents of a multimedia document as a method of enriching that document with additional valuable information, without altering the original content. The MADCOW (Multimedia Annotation of Digital Content Over the Web) system is a tool that enables its users to annotate contents of Web pages. MADCOW supports the annotation of (portions of) texts, images and videos with textual content, links to other resources, and user-defined tags [2]. Annotations can be published in MADCOW in one of three ways: (1) *public*: viewable by any user, (2) *private*: viewable by their submitters only, or (3) *group-related*: viewable by any

member of the group to which they are posted, and nobody else. Groups were introduced to MADCOW in [1] to compound the conflict between the use of public annotations, enhancing collaboration among users at the expenses of privacy, and private ones with the opposite characteristics. The introduction of groups solved this conflict by allowing sets of users to join a group focused on some common interest, and restricting publication of relevant annotations to this group. This allows a safer and more focused information exchange through annotations, fostering collaborative discussion threads on the group topics.

The creator of a new group can select one of three different policies to allow users to join this group: (1) **Public**: any user can join the group; (2) **Invite**: users can join a group only if invited by some group member with authoriser<sup>1</sup> privileges; (3) **Apply**: users apply for joining the group, and admission is subject to approval by one of the authorisers.

Authorisers deciding to invite users to their group and users looking for relevant groups share similar, if symmetrical, problems: “**how do I know who could be interested in joining my group?**” or “**what groups exist which might interest me?**”. Authorisers want to attract users interested in the group topic, in order to promote collaboration, while users want to share their thoughts with people who can provide interesting feedback. In both cases, they do not want to lose time with users and annotations not relevant to their interests.

In the current implementation of groups in the MADCOW system, authorisers and users do not have automated support to identify potentially interesting matches for their interests and have to manually search for them. Authorisers can list all MADCOW users (possibly looking at the annotations they submitted publicly) and select some of them as receivers of invitations. This process becomes rapidly unwieldy as the number of users and annotation increases. In a similar way, users can refer only to scant information about the subject of the group, namely the title and a textual description of the group topic, as they cannot access the content of the annotations posted to the group.

In an experimental test, 152 students from eight different undergraduate courses with their tutors were asked to use the system in order to check its feasibility and to measure a set of Human Computer Interaction metrics. The tutors of courses created different groups, all groups related services were tested and results were gathered. Table 1 presents information about the number of times the different operations on groups were used and the average time needed to complete the operation. It is clear that manual invitation takes the maximum time.

Table 1: Number and average duration (seconds) for executed operations.

	<i>Create</i>	<i>Update</i>	<i>Invite</i>	<i>Join</i>
<b># of times</b>	72	51	719	125
<b>Average</b>	37.3	15.9	99.25	5.6

<sup>1</sup>These are given to the group owner and to group moderators.

We argue that associating groups with publicly available representations of knowledge relevant to the group objectives, besides being a way to overcome these problems, provides additional advantages to the whole annotation activity. Indeed, we rely on the selection of well defined terms in which to express domain knowledge [4], which can both be recognised as significant by users, and automatically searched and manipulated by automated services [6, 8, 17, 18]. In particular, the endeavor of the Semantic Web has provided the community of programmers with ways to represent domain knowledge in the form of ontologies, typically represented using XML-based languages, such as RDF [13] or OWL [9].

In this sense, creators of groups will be able to associate them with existing ontologies, by browsing among the available ones. The process can be assisted by letting creators provide sets of terms which reflect the intent of the group. The MADCOW Ontology Browser will match these terms with the collections of terms associated with existing ontologies to propose those possibly appropriate for the new group.

On the annotator side, users can complement their annotations with tags. If they are members of some group, they can select the terms from the group ontology to better reflect the content of the annotation, facilitating subsequent retrieval. If users search for groups, they can equally submit terms who reflect their interests, and the MADCOW Ontology Browser will propose groups linked with ontologies including those terms. Analogously, users with authoriser privilege in some group can look for users whose public annotations contain terms in the group ontology.

This paper illustrates these concepts and describes the implementation of MADCOW Ontology Browser and its relation with the group-support mechanisms. The rest of the paper develops as follows: after considering related work in Section 2, we present the integration of ontologies in the MADCOW group annotation system in Section 3 and discuss some implementation issues in Section 4. Section 5 presents an applicative scenario and Section 6 concludes the paper.

## 2 Related work

There is a vast literature on ontologies and their applications in fields as diverse as education, medicine, software engineering, information retrieval, in order to increase interoperability between systems, classifications, matching domains or ensuring their consistency [16, 14].

Ontologies were proposed as representational schemes for domain knowledge to enhance the retrieval process by Paralic and Kostjal [10], who compared the efficiency of retrieval in an ontology-based approach (implemented in the Webocrat system) with the vector and the latent semantic indexing models, obtaining promising results. In [3] a retrieval agent is described that provides access to information from multiple domains based on domain ontologies and users' interests.

Patel *et al.* explored the use of ontologies in order to automate common clinical tasks, e.g. selection of a patient cohort for clinical trials, by considering the matching of patient records to clinical trials as a problem of semantic retrieval [12]. In their approach, clinical trial criteria are formulated as queries to be matched against a knowledge base (represented as an ontology) to retrieve eligible patients.

The work in [11] aimed at designing an ontology mapping algorithm for effective product matching between heterogeneous classifications of products. In electronic commerce, each shopping mall has its own vocabulary and product hierarchy, increasing the semantic interoperability problem. Their primary work is to minimize the number of retrieved products and to increase their relevance for customer searches.

Tangmunarunkit *et al.* matched resources to application demands [15], comparing attributes advertised by resources with those required by jobs. An ontology-based matchmaker was developed to perform resource selection by creating ontologies to declare resources and job requests, both expressed with RDF, and performing a semantic match between terms defined in the two ontologies. Similar to this, but in the area of human resource management, the work of [7] proposed an ontology-based approach to effectively match job seekers and job advertisements and applying a similarity-based approach to rank applicants.

Cantador *et al.* used representations of semantic user preferences for collaborative content retrieval, combining ontology-based user profiles to generate a shared semantic profile for a group of users was the work of [5]. They studied the feasibility of applying strategies for combining multiple individual preferences in a personalisation framework from a knowledge-based multimedia retrieval system. In their framework, user preferences are gathered in user profiles according to ontology concepts and used to retrieve ranked lists of items to be shown to users in graphical interface.

### 3 Relating ontologies and groups

We introduce some definitions preliminary to the description of the role of ontologies with respect to MADCOW groups.

1. **Domain:** a unique name designating the area of knowledge to which an ontology refers.
2. **Concept:** the name associated with a node in some ontology.
3. **Terms:** lexicalisations of the concepts in some ontology.
4. **Tags:** words (possibly terms in some ontology) provided by a user to characterise the annotation content.

The main idea for using ontologies in the process of matching groups with users is to find a collection of lexemes equivalent to the tags freely introduced by users as well as to the terms associated with some ontology. Based on the

correspondence ratio between words proposed by the user and the terms present in an ontology, a degree of matching between ontologies and annotations may be measured. Suitable thresholds can then be set on this degree of matching, to trigger the suggestion of potentially interesting groups to a user, or of potentially interested users to a group owner.

The following subsections describe how these matching processes are performed.

### 3.1 Group Creation

If an owner decides to link the new groups, several scenarios could take place. A group owner could provide the system with a possible name for a domain, and the system would check whether an ontology associated with this name exists. If such an ontology is found, an association is created. Otherwise, the owner is asked to provide the system with a set of terms which characterise the intent of the group and which might define its domain. If a significant number of these terms appear in some existing ontologies, these are presented to the user. Otherwise, the group creation process proceeds as normal.

The following pseudocode represents the previous steps:

```

group=askForGroupName();
if (referToDomain(group)) {
  domain=askForDomainName();
  if (existsDomainOntology(domain))
    associate(group, domain);
  else {
    terms=askForTerms();
    ontologyList=createEmptyList();
    foreach ontology in MADCOW.Ontologies {
      if (count(terms, ontology.getTerms())>=relevanceThreshold)
        ontologyList.add(ontology);
    }
    selectedOntologies=askForSelection(ontologyList);
    foreach ontology in selectedOntologies {
      associate(group, ontology.getDomain());
    }
  }
}

```

### 3.2 Annotation Submission

As we mentioned, users could submit their annotations as public, private or group-related. In the latter case, their tags can be construed as terms associated with the groups they are submitted to, and if the group is associated with an ontology, the tags have to be selected from the terms present in the ontology. For annotations submitted as public or private, submitters are free to use any set of tags as metadata associated with the annotation.

In general, tags attached to public or private annotations will be used in the matching process to propose groups to users, and users to group authorisers (in this case considering only public annotations).

The following pseudocode represents the annotation submission:

```
if (submitToGroup(annotation, group))
    assignTags(annotation, getGroupTerms(group));
elseif (attachTagsToAnnotation())
    assignTags(annotation, askForTags());
```

### 3.3 Matching Process

The main purpose for the matching feature implemented in the system, which is used on demand in a focused way, is to study the matching degree between annotations and domains. For groups' owners, matching means searching for candidate members while for users it means searching for candidate groups to which to apply for membership.

Having groups refer to domains reduces the time needed to execute the matching process because instead of executing the match between groups and annotations, the match is done between domains and annotations taking into account that each domain represents a set of groups which share the same terms used in the matching process. Involving only public (and possibly private) annotations for a user is another factor that minimizes the time consumed by the matching process.

In addition to results generated from the matching process, more results could be presented, based on the group owner's or user's needs:

1. For a group owner, check the presence of annotators who consistently annotate pages which are also annotated by writers of the group, and suggest those users to group owner.
2. For a user, check which groups have annotations on the same pages the user annotated, and suggest these groups to the user.

In case a group owner asks for potential members, the matching process executes a matching between the terms of the referring domain with the tags for all public annotations of users (not already members in the group) and proposes the most appropriate users. In case a user asks for potentially interesting groups, the matching process considers all of the user's annotations (public or private) with all domains existing in the system, and presents the groups related to the most significant domains. Fig. 1 depicts the matching process.

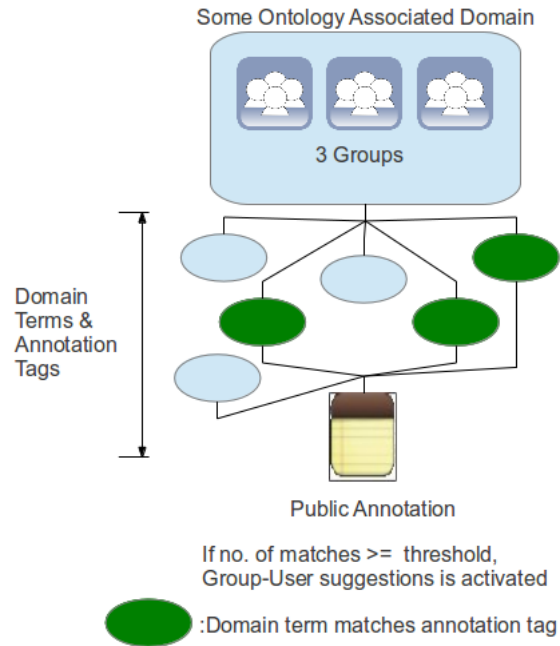


Fig. 1: Simple sketch illustrates the matching idea.

Given a domain  $D$  and an annotation  $A$ , let  $L_D = \{r_1, \dots, r_n\}$  be the set of lexicalisations of domain terms and let  $T = \{t_1, t_2, \dots, t_m\}$  be the set of tags. The matching between  $D$  and  $A$  is the process of checking the existence of a term from the set  $L_D$  in the set  $A$ , considering only exact matches. A significance threshold on the number of matches can then be used to filter out spurious domains.

In case this matching is executed by a group owner, the result will be a set of numbers representing the degrees of matching between the group and the public annotations for each MADCOW user (excluding those who are already writers in the group). When the matching is executed by a user, these numbers will represent the degree of matches between his/her annotations and all the domains existing in the system. A ranking feature could be introduced so that a group owner will get a descending sorted list of candidate writers according to their relevance to the group. The same feature could be introduced for a user asking for candidate groups, a descending order list of candidate groups presented to him/her.

Despite the simplicity of the proposed matching process, it introduces a seed for future works for using ontologies in such applications. More matching accuracy could be gained by implementing more accurate similarity measures.

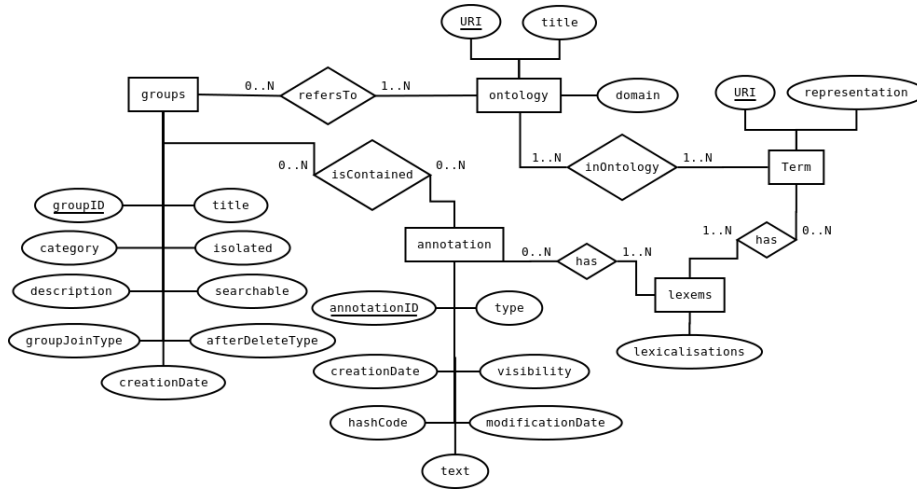


Fig. 2: A fragment of the MADCOW Entity-Relation diagram, considering groups, annotations, and ontologies.

## 4 System architecture and behavior

The following subsections describe the system architecture and behavior.

### 4.1 System Architecture

Fig. 2 presents a fragment of the overall Entity-Relationship diagram defining the logical scheme of the MADCOW database. We model here only the parts relevant to the management of ontologies and their association with groups.

The entity **groups** is used to save related data to each group created in the system with its properties. Each group created has a unique ID saved in **groupID**. A group also has a unique name saved in **title**, while **category** and **description** are used to save more details about the group. Typically, the first property is used in textual group search while the other is used for thematic search. When its property **isolated** is set to true, an annotation is readable only by its author (as well as by the group’s authorisers), and is not visible to other members. The way in which a group can be joined, i.e. whether users can request admission or need to be invited, depends on the properties **groupJoinType** and **searchable**. If a group is joined by **invitation**, then users cannot join it until they receive an invitation from one of the authorisers (even if they search for it), and if a group is joined upon **request**, then a user has to search for the group, list it, then send a joining request that will be handled by one of the group’s authorisers. When a group is deleted, its members can migrate to some other group, if the property **afterDeleteType** is set to “members in”, otherwise its members will return back as ordinary system users, and their related group annotations will become private ones. Finally,



the creation date for the group is saved in the property `creationDate`.

The entity `annotation` is used to store all the data relative to an annotation. A record is saved for each annotation submitted to the system by creating a unique ID in the field `annotationID`. The annotation type could be one of `Announcement`, `Comment`, `Example`, `Explanation`, `Integration`, `Memorandum`, `Question`, `Solution` or `Summary`, and is saved in the field `type`. The attribute `visibility` is used to state whether the annotation is `private`, `public` or `group`. Private annotations are viewed only by their submitters, public by all users, while group ones are just viewed by the members of the group where annotation submitter is member in. If an annotation is related to a group, a special record is saved in the proposed entity of the relation `isContained` that relates `annotation` and `groups` entities.

The link between a group and a suggested or chosen ontology is modeled as a relation `refersTo` that creates a relation between the entities `groups` and `ontology`. The latter has a property `domain` defining the area for which it has been defined. This value is referred to as a MADCOW domain. While the ER diagram indicates that a group could be referred to more than one domain, the current work is limited to considering the association only with a single domain, while a domain can be referred to by any number of different groups.

The property `title` of the `ontology` entity maintains the name of the ontology. The entity `Term` is related to the entity `ontology` by the relation `inOntology` so that an ontology has one or more terms. The concrete representation of the term is saved in the property `representation`, while the entity `lexemes` describes its possible lexicalisations. The same table is used to maintain tags used in annotations which do not have a direct reference to terms in a group domain ontology.

## 4.2 System Behavior

We illustrate the process of relating a group to an existing domain, possibly looking for an appropriate ontology. We also discuss the matching process between a group and candidate members, and between user's public and private annotations and ontologies. We present sequence diagrams describing the realisations of these use cases.

### 4.2.1 Group-Domain Relation

The process of relating a group to a domain is triggered when a group owner executes the activity `supplyDomainTitle(domain)` (see Fig. 3). By this function, the user is asking the system whether an ontology for a domain with this specific name is integrated in MADCOW. The system checks for the availability of the domain by executing the function `checkDomain(domain)`, querying the database with the name of the domain. If an ontology with that domain name is present, the system associates the group with that domain by executing the function `associate(group, domain)` that saves this association to the database. If there is no such a domain, the user is asked to provide

a set of terms to represent the aim for the group creation, via the function `provideTerms(terms[])`. When the user inputs the terms, the system executes the function `checkTerms(terms[])` that accesses the existing **Ontology Repository** to check whether there is some ontology containing the terms provided by the user. If some match is found, the retrieved ontologies are displayed for the group owner to choose one of them. If the user selects one of these ontologies, an association is created between the selected ontology and the group by executing the function `createAssociation(domain, ontology)`.

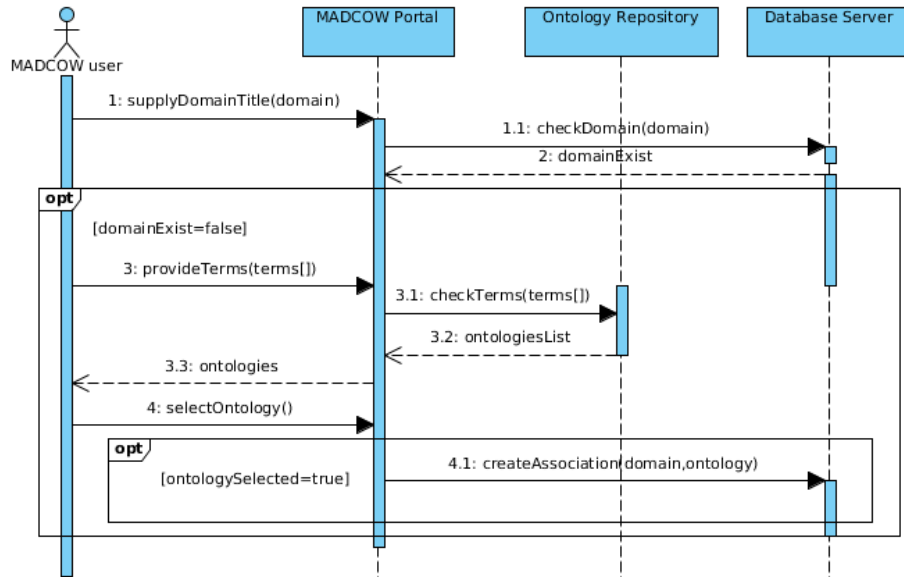


Fig. 3: Group-Domain Association Sequence Diagram.

#### 4.2.2 Suggestion of Members

A group owner could ask the system to suggest the most appropriate users to be candidate members in his/her group. As shown in Fig. 4, the process starts when the user executes the activity `suggestMembers(group)`. The system executes the function `matchWithAnnotations(group)` identified in the **MADCOW Portal**, which loads all the public annotations in the system database by executing the function `loadPubAnn()`. The **MADCOW Matcher** executes a matching process between tags in annotations and terms related to the domain associated with the selected group by executing the function `executeMatch(group, pubAnn)`. The output of this function is a ranked list of the most relevant users who could be candidate members for the group. Full details about users are retrieved from the system database to **MADCOW Matcher** by executing the function `loadMatchedUsers()`. The **MADCOW Matcher** sends the list of users to the **MADCOW Portal** via the function `showUsers()` and a list of users is presented to

the group owner for selection (`selectUsers()`). After the selection is completed, the group owner asks the system to send invitations for them by the activity `inviteUsers()`. Finally MADCOW Portal executes `sendInvitations(users,group)`, that saves user invitations to the database. Users will view these invitations when they log in to their accounts and can reply to accept or refuse.

### 4.2.3 Suggestion of Groups

Users can ask the system to suggest the most relevant groups exist in the system so that users could ask for memberships with these groups. The process (shown in Fig. 5) starts by a user request `suggestGroups()`, which makes the MADCOW Portal execute the function `matchAnnotationsWithDomains()` that makes the MADCOW Matcher to execute the function `loadDomains()` returning the existing domains. The MADCOW Matcher executes the function `loadPubPriAnn()` to load all the user's public and private annotations. The execution of `executeMatch(userAnnotations,domains)` produces a ranked list of all relevant domains with their associated groups. The activities `selectGroups()` and `joinRequests()` are performed by the user to activate the function `sendJoinRequests()` in the MADCOW Portal. This saves the user request to the database, to be viewed by group owners who can accept or refuse them.

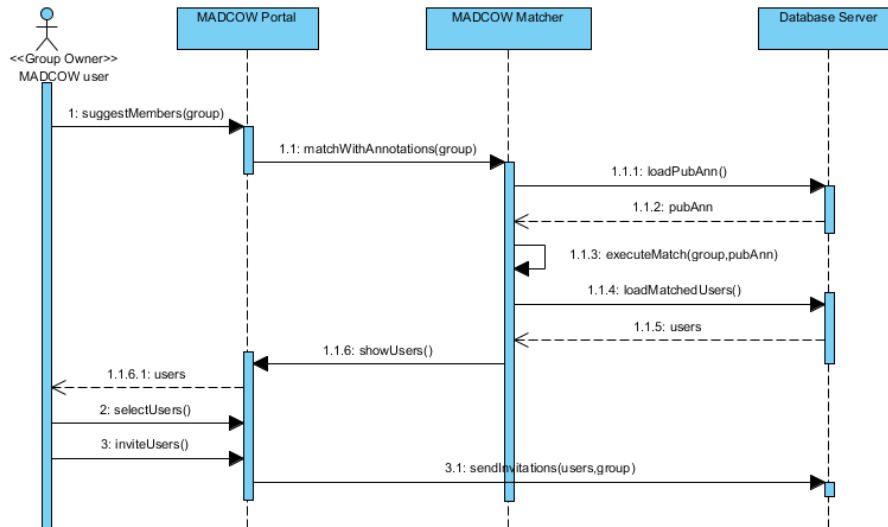


Fig. 4: Members Suggestion Sequence Diagram.

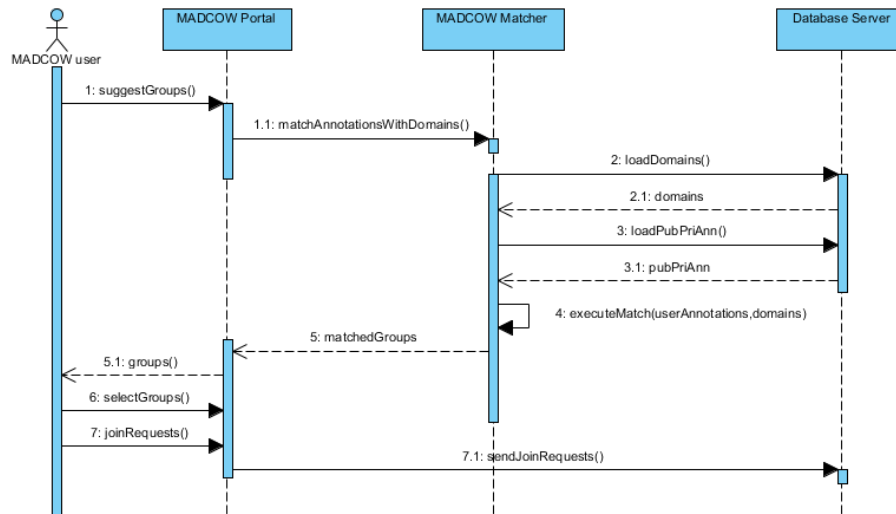



Fig. 5: Groups Suggestion Sequence Diagram.

## 5 Working Scenario

All the commands are executed from the MADCOW portal, reachable via the “Home” button, appearing in the Firefox toolbar once a user has logged in (see Fig. 6).

A university uses MADCOW annotation system as a coordination media for its faculty members and students. Aldo is a teacher who is interested in programming languages issues and he would like to create a group to gather members with this common interest. He creates a group titled “Java” and wants to profit from MADCOW ability to suggest members for his group. Hence, after creating the group from his account, he asks the system to browse the existing domains by clicking the  icon in front of the title of his new group, but finds out that no suitable domain exists. Fig. 7 shows a list of domains.

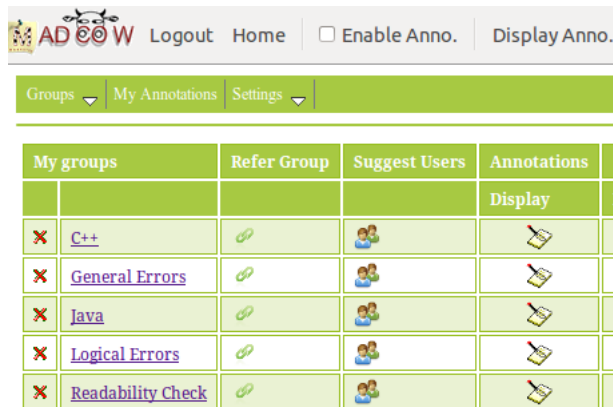


Fig. 6: MADCOW Toolbar.

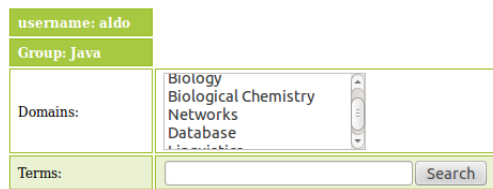



Fig. 7: Aldo lists available domains.

He then feeds his group with a set of proper terms such as “Structured”, “Object Oriented”, “Functional”, “High Level”, “JVM” to represent the intent of the group, and asks the system to suggest related ontologies for his group by clicking the button “Search”. The system uses the terms to search for the most suitable ontologies available and lists them for Aldo. Aldo picks the ontology with title “High Level Programming Languages” that has the following tags “Structured”, “Object Oriented”, “Multi-threaded”, “Simplicity” that appear when he selects the desired ontology. By clicking the button “Refer”, Aldo accepts the proposed ontology and his group is referred to the domain and the terms in the selected ontology become available for users to tag annotations with them. These steps are illustrated in Fig. 8.

username: aldo	
Group: Java	
Domains:	Biology Biological Chemistry Networks Database <input type="button" value="Select"/>
Terms:	ented, Functional, High Level, JVM <input type="button" value="Search"/>
Proposed Ontologies:	Programming Languages Programming with C++ <b>High Level Programming Languages</b> Object Oriented Programming Languages
Tags:	Structured Object Oiented Multi-threaded Simplicity <input type="button" value="Refer"/>

Fig. 8: Aldo supplies the system with set of terms and selects “High Level Programming Languages” ontology.

Aldo asks the system to suggest members for his new group by clicking the  icon in front of the group title, and the system looks for matches between the domain terms and all tags of public annotations. Aldo is then presented with a ranked list of users. He selects them all to send invitations to the group. Fig. 9 illustrates the invitation process.

Group: Java		
Relevant Users:	username	Rank
<input checked="" type="checkbox"/>	crisrina	92%
<input checked="" type="checkbox"/>	taya	87%
<input checked="" type="checkbox"/>	alessandro	85%
<input checked="" type="checkbox"/>	dando	80%
<input checked="" type="checkbox"/>	edwardo	80%
<input checked="" type="checkbox"/>	daniel	70%
<input type="button" value="Invite"/>		

Fig. 9: Aldo sends invitations to most relevant users.

## 6 Conclusions and future work

We discuss the use of ontologies in the MADCOW annotation system to complement the notion of group, and present an approach to find matches between groups and potentially interested users. Ontologies are used to represent domain knowledge relevant to the formation of MADCOW groups. Each domain with its associated ontology can provide terms, used as tags, for annotations to be published in groups related to that domain, facilitating a better structuring of the knowledge shared within the group. Tools for facilitating the retrieval of interested users (or of interesting groups) are provided based on matches be-

tween tags freely used by submitters of annotations and terms contained in the different ontologies integrated in the MADCOW system.

## References

- [1] Danilo Avola, Paolo Bottoni, Marco Laureti, Stefano Levialdi, and Emanuele Panizzi. Managing groups and group annotations in madcow. In *Proc. DNIS 2010*, volume 5999 of *LNCS*, pages 194–209, 2010.
- [2] Paolo Bottoni, Roberta Civica, Stefano Levialdi, Laura Orso, Emanuele Panizzi, and Rosa Trinchese. Madcow: a multimedia digital annotation system. In *Proc. AVI'04*, pages 55–62. ACM, 2004.
- [3] R.M.M. Braga, C.M.L. Werner, and M. Mattoso. Using ontologies for domain information retrieval. In *Proc. DEXA 2000*, pages 836–840, 2000.
- [4] Christopher Brewster, Kieron O'Hara, Steve Fuller, Yorick Wilks, Enrico Franconi, Mark A. Musen, Jeremy Ellman, and Simon Buckingham Shum. Knowledge representation with ontologies: The present and future. *IEEE Intelligent Systems*, pages 72–81, January 2004.
- [5] Ivan Cantador, Pablo Castells, David Vallet, and Escuela Politcnica. Enriching group profiles with ontologies for knowledge-driven collaborative content retrieval. In *Proc. STICA 2006 at WETICE 2006*, pages 358–363, 2006.
- [6] B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, January 1999.
- [7] Maryam Fazel-Zarandi and MarkS. Fox. Reasoning about skills and competencies. In LuisM. Camarinha-Matos, Xavier Boucher, and Hamideh Afsarmanesh, editors, *Collaborative Networks for a Sustainable World*, volume 336 of *IFIP Advances in Information and Communication Technology*, pages 372–379. Springer Berlin Heidelberg, 2010.
- [8] Richard Gil, Ana Maria Borges, and Leonardo Contreras. Shared ontologies to increase systems interoperability in university institutions. In *Proc. IMCSIT 2007*, pages 799–808, 2007.
- [9] OWL Working Group. OWL Web Ontology Language. Technical report, OMG, 2004.
- [10] Jan Paralic and Ivan Kostial. Ontology-based information retrieval. *INFORMATION AND INTELLIGENT SYSTEMS, CROATIA*, pages 23–28, 2003.

- [11] Sangun Park, Wooju Kim, Sunghwan Lee, and Siri Bang. Product matching through ontology mapping in comparison shopping. In *Proc. iiWAS*, volume 214 of *books@ocg.at*, pages 39–49. Austrian Computer Society, 2006.
- [12] Chintan Patel, James Cimino, Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Li Ma, Edith Schonberg, and Kavitha Srinivas. Matching patient records to clinical trials using ontologies. In *Proc. ISWC'07/ASWC'07*, pages 816–829, Berlin, Heidelberg, 2007. Springer-Verlag.
- [13] RDF Working Group. RDF/XML Syntax Specification (Revised). Technical report, OMG, 2004.
- [14] Stefan Seedorf, Fzi Forschungszentrum Informatik, and Universitt Mannheim. Applications of ontologies in software engineering. In *In 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), held at the 5th International Semantic Web Conference (ISWC 2006)*, 2006.
- [15] Hongsuda Tangmunarunkit, Stefan Decker, and Carl Kesselman. Ontology-based resource matching in the grid - the grid meets the semantic web. In *Proc. ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 706–721, 2003.
- [16] Mike Uschold, Michael Gruninger, Mike Uschold, and Michael Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–136, 1996.
- [17] David Vallet, Miriam Fernandez, and Pablo Castells. An ontology-based information retrieval model. In *Proc. ESWC 2005*, pages 455–470. Springer, 2005.
- [18] Yi Zhang, Wamberto Vasconcelos, and Derek Sleeman. Ontosearch: An ontology search engine. In Max Bramer, Frans Coenen, and Tony Allen, editors, *Research and Development in Intelligent Systems XXI*, pages 58–69. Springer London, 2005.