User Profile Elicitation and Conversion in a Mashup Environment

Erwin Leonardi[1], Geert-Jan Houben[1,2], Kees van der Sluijs[2], Jan Hidders[1], Eelco Herder[3], Fabian Abel[3], Daniel Krause[3], Dominikus Heckmann[4]

[1] Delft University of Technology, PO Box 5031, 2600 GA Delft, the Netherlands
{e.leonardi, g.j.p.m.houben, a.j.h.hidders}@tudelft.nl
[2] Technische Universiteit Eindhoven, PO Box 513, 5600 MB, Eindhoven, the Netherlands
k.a.m.sluijs@tue.nl
[3] L3S Research Center, Appelstr. 9a, 30167 Hannover, Germany
{herder, abel, krause}@l3s.de
[4] German Research Center for Artificial Intelligence, Saarbrucken, Germany
heckmann@dfki.de

Abstract. Many Web applications have offered personalization and adaptation as their features in order to provide personalized services to their users. The user profiles are gathered independently by these applications often through an explicit dialogue with the user. As a result, the users have to go through a similar elicitation process multiple times, that is, providing similar information that is used to build the user profiles to different applications. In the springtime of mashup applications, we observe the importance of considering user information in order to make the presented content more relevant to the user. For this purpose, it is necessary to have a platform/framework that enables components in the mashup to reuse and exchange user profiles. In this paper, we present the Morpho framework that elicits, enhances, and transforms a user profile from one application to another application in a mashup environment. It deals with *semantic* and *syntactic heterogeneity* of data and schema of the user profile. We present the architecture of Morpho and a case study to exemplify the approach followed in this current work.

Keywords: user profile, interoperability, mashup

1 Introduction

With the evolution of the Web, Web applications have become more complex and offer their users many interesting and advanced features. Adaptation and personalization become important features of today's Web applications. In order to be able to adapt and offer personalized contents for a specific user, a Web application must have enough information about this user. This information can be gathered explicitly and implicitly. The explicit approach is by asking directly to the user, for example, by using a survey form or by asking the user to give ratings to certain products, thus building up a user profile. In the latter approach, the Web application monitors the behaviors of the users while the users use the application in order to construct a user model fitting with the goal of the application.

Parallel to the growing of the Web, the number of Web applications offering various services has also increased significantly. Consequently, Web users may be using more than one application that offer different products/services, and keep adding new applications. Each of these applications independently asks for the user profiles of its users that are used to provide personalized contents and services. They do not share or learn from one another. This leaves no choice for the users but to go through the same process again and again. That is, the users have to, for example, fill in different questionnaires for different applications or rate products until the applications have enough information to describe their interests. This can be a cumbersome thing to do for many users and something for which support to help them in constructing these profiles is welcome.

Nowadays a new breed of Web applications called *mashup* has been deployed in several areas and increasingly becomes more popular. A mashup combines data from two or more sources into a single integrated tool. The data that originally belongs to other Web applications is blended in order to provide enriched user-oriented contents. Note that the Web applications from which a mashup tool fetches data may have their own specific ways to describe information about their users. We suggest that by considering information about a user we can have a better mashup - a mashup that provides *more relevant* contents for the user. In addition, the sharing of user profiles facilitates a better integration and cooperation between underlying applications in the mashup. For example, consider a mashup Web application named BookTour[1] that combines Google Earth[2] and Amazon[3] to show book tours happening around the world. It would be more relevant and interesting for a user if BookTour shows only the events that are related to, for example, favorite authors of this user based on the books in Amazon that he/she buys and rates. So, BookTour uses what it knows about the user to ask more specific queries to Amazon resulting in more relevant information from Amazon. Observe that a user may only use several applications provided in a mashup, but never uses the rest of the available application. This causes the absence of a user profile in some of these applications. Consequently, these applications may not be able to provide customized and personalized contents for this user. The challenge is to develop a mechanism to reuse the information about a user, which is originally from one application, for another application [1,2]. We refer to this as *user profile interoperability*.

One technique for user profile interoperability is by using a unified (central) model that serves as a predefined structure and is easily exchangeable and interpretable [1,11]. However, it is impractical to force Web applications to use a unified model because the Web is an open and dynamic environment [3]. A more flexible alternative approach is to provide a framework that elicits the data in the user profile of one application and *transforms* it into a user profile of another application. Thus the mashup can use the profiles to retrieve more relevant content from the underlying applications. If we do this integration by exploiting Semantic Web techniques, it can be flexible because the applications do not need to follow a fixed model for their user profile. This technique raises some main challenges, that is, to deal with *semantic* and

[1] http://www.booktour.com/
[2] http://earth.google.com/
[3] http://www.amazon.com/

syntactic heterogeneity of data and schema of the user profile. The flexibility offered by this approach has a trade-off. The main advantage is that the "glue" that the mashup represents can be applied more precisely and that the connection between mashup and base applications works at an increased relevance level. However, in many situations (especially when starting the use of an application at the-so-called cold start [20]) an increase in relevance is welcome. Of course, a transformation of profiles will not be perfect as it may lead to the possibility of losing data during transformation process. It is also possible that the transformation cannot be made because the model is simply incompatible.

In this paper, we propose a framework (called Morpho) that aims at eliciting the user profile of an application and transforming it to the one of another application. It is a part of our *User Pipes* project that aims to allow user profile reasoning by mashing up different user profile data streams. It can be used as a component in a mashup application to ensure user profile interoperability and sharing between other mashup components. It deals with semantic and syntactic heterogeneity of user profiles and ensures the interoperability of user profiles of different applications. In addition, it is configurable and extensible as the mashup application administrators (the *administrators*) are able to specify a configuration by which elicitation and transformation processes are guided. Note that this framework works with the assumption that user profile data can only be accessed and exchanged with explicit consent of the owner. This assumption is important because the privacy issue is critical for user profiles and modeling [4]. As Morpho can be used as a component in a mashup application, the explicit consent given by the mashup application users (the *end-users*) to the mashup application is also given to Morpho.

This paper reports on our ongoing work related to Morpho and User Pipes and it is structured as follows. In Section 2, we present related work. Section 3 presents a motivating case study that is used as running example in the discussion. Section 4 discusses the architecture of the Morpho framework and elaborates on user profile elicitation and transformation. Finally, Section 5 concludes the paper by highlighting some future directions of this research.

2 Related Work

To address the user profile interoperability there are basically two approaches: the *shared format* approach and the *conversion* approach. In the shared format approach, a common language for a unified user profile (*a lingua franca*) is needed. Applications have to follow the unified format [13]. Examples of this approach are the General User Model Ontology (GUMO) [1] and Composite Capability/ Preference Profiles (CC/PP)[4]. This approach is easily exchangeable and interpretable as there is no syntactic and semantic heterogeneity issue to be addressed [1]. However, this approach is not suitable for open and dynamic environments, such as the Web, as it is impractical and in many cases impossible to enforce Web applications to follow the *lingua franca* [3]. The conversion approach is more flexible and suitable for open and

[4] http://www.w3.org/Mobile/CCPP/

dynamic environments [14]. In this approach, a technique has to be developed for converting a user profile of one application to another application. The developed technique should deal with the problem of syntactic and semantic heterogeneity. Observe that potential drawbacks of this approach are that it is possible that some information is lost during the conversion process, and that it is possible that models are simply incompatible. This means that there is no suitable mapping for these models. It is also possible that the mappings are incomplete because required information in one model is not available in the other model.

Mashing up data and tools into one integrated tool has become increasingly popular recently [15, 16, 17]. One work in the mashup related environment that is closely related to our research is presented in [18, 19]. In [18,19], Gosh et al. present a framework called SUPER (*Semantic User Profile Management Framework*) for capturing and maintaining user profiles using semantic web technology in the retail domain. SUPER aggregates user profile information that spreads over several services/data sources.

There is a host of related work about user profiling and about mashup that could be discussed; however, we do not discuss this in this workshop paper.

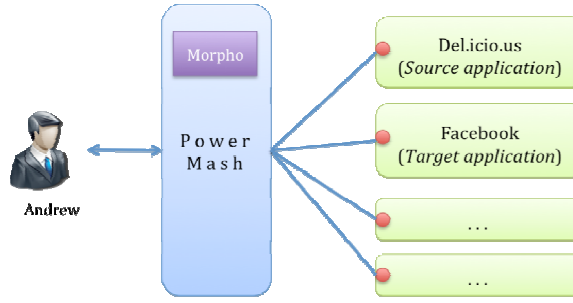

Figure 1: The PowerMash Application

3 Motivating Case Study

In this section, we present a case study to exemplify our approach discussed in the subsequent sections. Consider an end-user named Andrew that uses a mashup application called PowerMash as shown in Figure 1. Two of the underlying applications are Del.ious.us[5] and Facebook[6]. Suppose Andrew has used Del.icio.us to bookmark web pages that are interesting for him. He has a Facebook account; however, he has not filled up many parts of his Facebook profile. PowerMash uses the Morpho framework as its component for user profile elicitation and transformation. In this scenario, the challenge is to see to what extent Morpho is able to use Andrew's

[5] http://delicious.com/
[6] http://www.facebook.com/

bookmark entries in Del.icio.us to help him enhance his Facebook profile as much as possible.

Figure 2 depicts three bookmark items of Andrew. Each bookmark item in Del.icio.us contains fields of information about creation date, URL, title, note/description, and tags for the bookmark item. The URL field identifies the resource location (or web address) of a bookmarked web page. The title field stores the title of a bookmarked web page. A user may write additional description or information about the web page. This information is maintained in the note field. Tags are typically one-word descriptors that a user can assign to his/her bookmarks to help him/her organize and remember them. Each bookmark item can have many tags and they do not form a hierarchy. Note that the note and tags are optional. Consider the third bookmark item shown in Figure 2. This bookmark item was created on 8 April 2009 and has URL `www.youtube.com/watch?v=73-V2A3NuWo` and title "*Kelly Clarkson – Because of You*". Andrew has put a note "*High quality video from official Kelly Clarkson channel at YouTube*" and given the tags "*pop"* and *"song*".

For the purpose of enhancing and improving Andrew's Facebook profile (as much as possible), PowerMash employs Morpho to elicit Andrew's user profile in the form of bookmark entries in Del.icio.us and transform them into elements of his Facebook profile. In this situation, PowerMash retrieves Andrew's bookmark entries from Del.icio.us. Note that the communications between a mashup application and its underlying basae application can be performed by using API, RSS feeds, RESTful service, or SOAP service. Since Morpho is for user profile elicitation and transformation, the way a mashup application retrieves and sends data to its underlying applications or web services is beyond the scope of our work in this paper. Having retrieved Andrew's bookmark entries, PowerMash sends a request to Morpho to transform his bookmark entries into data for his Facebook profile.

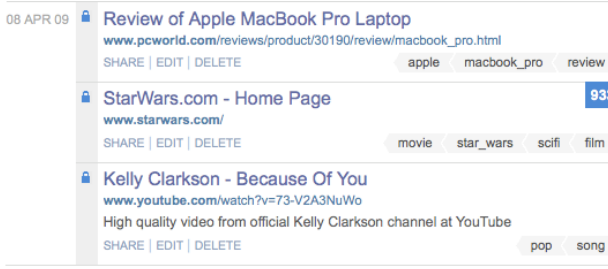

Figure 2: Three Bookmark Items of Andrew from Del.icio.us

4 Morpho Framework

Figure 3 depicts the architecture of the Morpho framework. Let us first give the overview of the modules in the framework. We will elaborate them in details in the subsequence sections. As the name suggests, the *Interface* module manages the interactions between the framework and a mashup application, in our case, PowerMash. It allows the mashup to submit inputs to the framework and to receive

the converted user profiles produced by the framework. Next to the Interface is the *Controller* module that serves as the main controller of the framework. The *Model Builder* module processes the source user profile (in our case, Andrew's bookmarks) and maps it into a corresponding internal conceptual model using a set of mapping rules. Then, it generates RDF based on this conceptual model, and stores it in the *User Profile Repository*. A set of concepts has to be extracted from the source user profile. Extraction is necessary because the content of a user profile is typically text that is *meaningless* for machine. The *Concept Extractor* performs this task by using available knowledge bases, in our case, DBpedia. The conversion of a source user profile to a target user profile takes place inside the *Interoperability Engine*. This engine measures the distance between the concepts in the source user profile and the concepts related to target user profile schema. Finally, the *Result Builder* generates the user profile of the target application. To simplify the discussion in our running example we consider only to fill up the '*favorite music'* field of the Facebook profile.

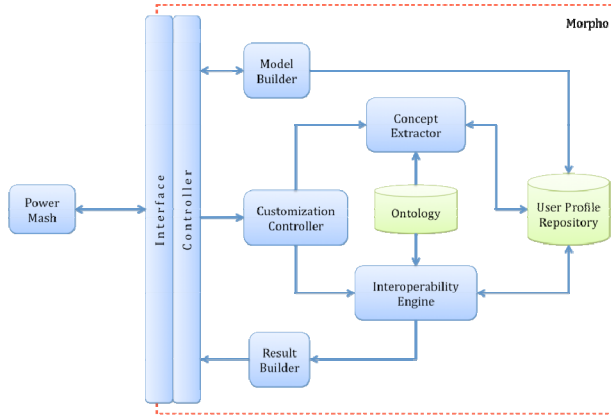

Figure 3: The Architecture of Morpho

4.1 Controller

After the mashup application sends the necessary input to the framework via the Interface, the Controller starts its task of managing the elicitation and transformation of user profiles. All inputs received from the mashup, except the configuration, are fed to the Model Builder. The configuration is sent to the Customization Controller. In some cases, the source and target applications are from the same application domain. For example, the source and target applications can be both social network sites. If the Controller detects such cases, several steps can be skipped. Let us elaborate further on this. Suppose the source application is Facebook and the target application is Hyves[7]. A set of mapping rules transforms a profile from Facebook into RDF based on Morpho internal conceptual model for social network sites. Then, this RDF is mapped into Hyves profile using another set of mapping rules that is

[7] http://www.hyves.nl/

specifically used to transform Hyves profile into Morpho internal conceptual model for social network sites and vice versa. Observe that even though two applications from the same application domain share the same Morpho internal conceptual models, the user profile transformation among them will not be perfect as during transformation process it is possible to lose data. It is also possible that required information in target profile is not available in the source profile.

4.2 Model Builder

The main task of Model Builder is to parse and map the source user profile to a specific internal conceptual model in Morpho, and to transform this user profile into RDF based on this internal conceptual model. The mapping functionality is application specific. This means that for a particular Web application a set of *mapping rules* has to be defined. The internal conceptual model is domain specific. We use the same internal conceptual model for all applications that belong to the same application domain. For example, if the source application is a social network site, then the internal conceptual model can be based on the OpenSocial/RDF[8] and FOAF specification[9]. Another example is that for bookmark data, we use our internal bookmark conceptual model that is inspired by the Annotea Bookmark Schema[10]. In our running example, the Model Builder transforms Andrew's bookmarks depicted in Figure 2 into RDF as follows:

```
<rdf:RDF ... >
  <morpho:Bookmarks>
    <morpho:hasItem rdf:resource="Bookmark_1" />
    ...
  </morpho:Bookmarks>
  <morpho:Bookmark rdf:ID="Bookmark_1">
    <a:created>2009-04-08T09:54:49+0100</a:created>
    <dc:title>Kelly Clarkson - Because Of You</dc:title>
    <dc:description>High quality video from official Kelly
         Clarkson channel at YouTube</dc:description>
    <b:recalls>http://www.youtube.com/watch?v=73-V2A3NuWo</b:recalls>
    <a:hasAnnotation>pop</a:hasAnnotation>
    <a:hasAnnotation>song</a:hasAnnotation>
  </morhpo:Bookmark>
  ...
</rdf:RDF>
```

In addition, the Model Builder also builds an RDF model for the target user profile based on the target application and annotates it with default pre-defined related concepts. For instance, the property '*music*' in the social network conceptual model that describes the user's favorite music is annotated with the DBpedia concept labeled '*music*' (e.g. dbpedia:Category:Music). This RDF model acts as the schema for the target user profile. Its instances are generated from the source user profile. The generated RDF is stored in an RDF repository (e.g. Jena [6] and Sesame [7]).

[8] http://web-semantics.org/ns/opensocial
[9] http://xmlns.com/foaf/spec/
[10] http://www.w3.org/2001/Annotea/

4.3 Customization Controller

Our framework employs various tools or external components that perform certain tasks in our modules, namely, in the Concept Extractor and Interoperability Engine modules. We shall discuss how these tools are utilized in the subsequent sections. One of the important features that we have is to allow the administrators to be able to guide how some modules should work by specifying the preferable combination of tools and components. For example, an administrator may prefer to aggregate the similarity scores that are returned by Levenshtein Distance [7] and Soundex [8] for the lexical matching step discussed in subsequent section. He/she might also want to specify, for example, which properties should be followed while expanding concepts. The Customization Controller takes the preferences and feeds the Concept Extractor and Interoperability Engine modules with necessary settings. The administrators do not specify their preferences, and then a default configuration is used.

4.4 Concept Extractor

The RDF data stored in the RDF repository is still *raw* and cannot be used directly to generate the target user profile. We need to connect this data to the concepts, in our case, the DBpedia concepts, such that it is meaningful for the machine. The Concept Extractor has to determine a set of concepts out of the RDF of the source user profile, in this case, Andrew's bookmarks. Note that this module is inspired by Relco [9], a tool for relating tags to concepts. The Concept Extractor works as follows.

Firstly, the Concept Extractor finds a set of keywords from the information available in Andrew's bookmark items. The keywords can be discovered using two tools for natural language processing, namely, Part-Of-Speech Tagger (POS Tagger) [10] and Named Entity Recognition (NER) [11] tools. A POS tagger is a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc., although generally computational applications use more fine-grained POS tags like 'noun-plural'. NER labels sequences of words in a text, which are the names of things, such as person and company names. It facilitates the framework to determine, for example, the person name. We consider words that are detected as noun and preceded by zero or more adjectives as the keywords. In addition, one or more words, which are determined as the entity names, are also considered as a keyword. For example, the POS tagger assigns parts of speech to each word in the description of the third bookmark item of Andrew as follows:

```
High quality video from official Kelly Clarkson channel at YouTube
 JJ    NN      NN   IN    NN       NN    NN       NN    IN   NN
```

There are seven keyword candidates: '*high quality*', '*video*', '*official*', '*Kelly*', '*Clarkson*', '*channel*', and '*YouTube*'. The NER returns '*Kelly Clarkson*' and '*YouTube*' as possible entity names. Combining the results of these tools, we have six keywords: '*high quality*', '*video*', '*official*', '*Kelly Clarkson*', '*channel*', and '*YouTube*'. Note that another sets of keywords are extracted from URL, title, and tags

of the bookmark items. We use the instance of the *morpho:Keyword* class to describe the discovered keywords:

```
...
<morpho:Bookmark rdf:ID="Bookmark_1">
   ...
   <morpho:Keyword ID="Keyword_1">
      <morpho:keywordValue>high quality</morpho:keywordValue>
   </morpho:hasKeyword>
   <morpho:Keyword ID="Keyword_2">
      <morpho:keywordValue>video</morpho:keywordValue>
   </morpho:hasKeyword>
   ...
```

The next step is to *lexically* match the discovered keywords from Andrew's bookmarks to a set of candidate concepts from DBpedia. A concept in DBpedia is of type *skos:concept* and usually has a property that describes the label of this concept, such as *rdfs*:*label*. In some cases, a concept can have multiple labels denoted by, for example, *skos*:*prefLabel* and *skos*:*altLabel*. These properties are in the SKOS vocabulary[11]. These matches give us a set of candidate concepts that may be syntactically related to the keywords together with their similarity scores. For example, in our running example, the DBpedia concept labeled '*Video'*[12] could be a good matching for the keyword '*video'*. Observe that the labels that are encoded in URIs are also considered [9]. In our implementation, the open source SimMetrics library[13] is used. This library employs many well-known similarity metrics such as Levenshtein Distance [7], Soundex [8], etc. The administrators can choose which metrics they want to use and how they are aggregated. The instance of *morpho*:*RelatedConcept* is used to describe the DBpedia concepts that are related to the keywords:

```
...
<morpho:Keyword ID="Keyword_2">
   <morpho:keywordValue>video</morpho:keywordValue>
   <morpho:RelatedConcept>
      <morpho:extConcept rdf:resource="DBpedia:Category:Video" />
      <morpho:similarityScore>1.0</morpho:similarityScore>
   </morpho:relatedToConcept>
   ...
```

The third step is to find other DBpedia concepts that are *semantically* related to the DBpedia concepts discovered in the previous step. This can be done by following and exploiting some properties of the concepts, for example, rdfs:subClassOf, *skos*:*related*, or *skos*:*broader*. The administrator can configure additional properties that the framework should follow during semantic structure exploitation. In our running example, the concept labeled '*Video'* is semantically related to the concepts labeled '*Music and video'* and '*Film'*. Thus, they can also be related to the keyword '*video'* extracted from Andrew's bookmarks. These found related concepts can be useful and might be a good alternative to the original concepts [9]. The *morpho*:*relatedToConcept* is also used to the new discovered concepts. The similarity

[11] http://www.w3.org/TR/skos-reference/
[12] http://dbpedia.org/page/Category:Video
[13] http://sourceforge.net/projects/simmetrics/

score of these new discovered concepts are based on the similarity score of the original concept, but lowered by a configurable reduction factor.

The previous step might result many related concepts for each keyword. These related concepts are ranked according to their similarity scores; however, by knowing the properties of the source user profile, the related concepts can be processed and refined further. In our running example, we have a set of bookmark items of Andrew's Del.icio.us bookmarks. Each bookmark item maintains information (e.g. URL, title, etc.) about *one* bookmarked website. By knowing this, we are able to disambiguate and to better select concepts that are most appropriate for each bookmark item. Intuitively, the probability is high that the keywords from a bookmark item will relate to concepts that are closed to each other. Consider the third bookmark item in Figure 2. For instance, the keyword '*video*' is related to DBpedia concepts labeled '*Video'*, '*Music and video'*, and '*Film'*. However, if we consider another keyword from the third bookmark item, for example, keywords '*pop*' and '*song*', then the concept labeled '*Music and video'* is more appropriate and relevant for the third bookmark. Observe that the keywords '*pop*' and '*song*' are related to the DBpedia concepts labeled '*Pop music*' and '*Songs*', respectively. These concepts are closer to the concept labeled '*Music and video'* than to the ones labeled '*Video'* and '*Film'*. Note that maximum distance between concepts to be considered as close to each other is configurable by the mashup application administrators.

4.5 Interoperability Engine

Section 4.4 discussed how the concepts are extracted from the source user profile (e.g. Andrew's Del.icio.us bookmarks). In this section, we elaborate on how these extracted concepts are related to the annotated DBpedia concepts in the conceptual model of the *target* user profile (e.g. DBpedia concept labeled '*Music*' that describes favorite music properties in the Morpho internal conceptual model of Facebook).

The Interoperability Engine measures the distance between the concepts that are extracted from the source user profile and the annotated DBpedia concepts in the conceptual model of the *target* user profile. To measure the similarity and relatedness, we employ DBpedia Relationship Finder [12] that is able to compute the distance between two objects/concepts in DBpedia. In our running example, the Interoperability Engine computes the distances between the DBpedia concepts related to Andrew's bookmark items and the DBpedia concepts labeled '*Music*'. The first bookmark item in Figure 2 is related to the DBpedia concept labeled '*Laptops*'. Recall that the third bookmark item is related to the DBpedia concept labeled '*Pop music*'. The path from the concept labeled '*Laptops*' to the concept labeled '*Music*' is much longer than the path from the concept labeled '*Pop music*' to the concept labeled '*Music*' and therefore the concept labeled '*Laptops*' is considered not relevant for the concept labeled '*Music*'. Then, the Interoperability Engine establishes link between the concept labeled '*Pop music*' and the concept labeled '*Music*' using a property *morpho:isRelevantTo*.

4.6 Result Builder

In the previous step, a set of concepts that is semantically related to the concepts in the target schema has been determined. The Result Builder exploits the *morpho*: *isRelevantTo* property and maps the target user profile (based on our internal conceptual model) to the one of the target application using a set of defined mapping rules. In our running example, the concepts '*Kelly Clarkson'*, '*music video'*, and '*pop music'* are relevant for the '*favorite music*' in Facebook and can be used to enhance it.

5 Conclusions and Future Work

In this paper we emphasize the importance of sharing and exchanging user profiles between underlying applications in personalized mashup applications. We suggest that by considering and sharing information about a user we can have a better mashup - a mashup that provides *more relevant* contents for the user. In addition, the sharing of user profiles facilitates a better integration and cooperation between applications in the mashup. We also propose a framework called Morpho that is a part of our *User Pipes* project. It is used to elicit a user profile from an application, and transform it to a profile for another application. It can be employed as a component in a mashup application and helps the mashup to perform user profile interoperability. Also, it is extensible and configurable as the mashup application administrator is able to specify settings that guide some processes in Morpho.

In the e-learning domain, in the context of the GRAPPLE project we are also working on a framework called GUMF (Grapple User Model Framework) for exchanging user model of various e-learning systems. The idea behind GUMF and Morpho is similar that is to enable user profile interoperability of various applications. However, GUMF is specifically designed and configured for the e-learning domain, while we intend to make Morpho applicable for various domains in which lightweight composition or mashups are relevant.

Even though the Morpho framework is able to provide the basic framework for user profile elicitation and transformation in the mashup environment, its performance depends on the algorithms/tools that are employed. Further evaluation and experimentation of the framework has to be done in order to see to what extend-user profile elicitation and transformation can be done. In addition, the evaluation has to study additional requirements and further extensions of Morpho. Our ongoing research on *User Pipes* also helps us in extending the functionality and portability of this proposed framework. For example, we can observe how WordNet can also be exploited to determine the semantic relatedness between two terms. Consequently, this can be used as an alternative of computing the semantic distance performed by the Interoperability Engine. Similarly, using other kinds of natural language tools in the Concept Extractor can be beneficial.

Acknowledgements: This work was partially supported by the European 7th Framework Program project GRAPPLE ('Generic Responsive Adaptive Personalized Learning Environment').

References

[1] D. Heckmann, T. Schwartz, B. Brandherm, M. Schmitz, and M. von Wilamowitz-Moellendorff. GUMO - The General User Model Ontology. In Proc. of 10th International Conference on User Modeling (UM 2005), Edinburgh, UK, Jul, 2005.

[2] F. Cena and L. Aroyo. A Semantics-Based Dialogue for Interoperability of User-Adaptive Systems in a Ubiquitous Environment. In Proc. of 11th International Conference on User Modeling (UM 2007), Corfu, Greece, Jun 2007.

[3] T. Kuflik. Semantically-Enhanced User Models Mediation: Research Agenda. In Proc. of 5th International Workshop on Ubiquitous User Modeling (UbiqUM'2008), workshop at IUI 2008, Gran Canaria, Spain, Jan, 2008.

[4] A. Kobsa. User Modeling in Dialog Systems: Potentials and Hazards. In AI and Society, 4(3):214-240, 1990.

[5] B. McBride. Jena: A Semantic Web Toolkit. In IEEE Internet Computing, 6(6):55–59, 2002.

[6] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Proc. of International Semantic Web Conference (ISWC 2002), Sardinia, Italy, Jun, 2002.

[7] F. Damerau. A Technique for Computer Detection and Correction of Spelling Errors. In Communications of the ACM, ACM, vol. 7, no. 3 (1964), 171-176.

[8] D. E. Knuth. The Art of Computer Programming Volume 3: Sorting and Searching. Addison-Wesley (1973), 394-395

[9] K. van der Sluijs and G.-J. Houben. Relating User Tags to Ontological Information. Proc. of 5th International Workshop on Ubiquitous User Modeling (UbiqUM'2008), workshop at IUI 2008, Gran Canaria, Spain, Jan, 2008.

[10] K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proc. of HLT-NAACL, 2003, 252-259.

[11] J. R. Finkel, T. Grenager, and C. Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In Proc. of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005), USA, 2005.

[12] J. Lehmann, J. Schüppel, S. Auer. Discovering Unknown Connections – the DBpedia Relationship Finder. In Proc. of 1st Conference on Social Semantic Web, CSSW2007, Leipzig, Sep 24–28, 2007.

[13] C. Stewart, A. Cristea , I. Celik , and H. Ashman. Interoperability between AEH user models. In Proc. of the Joint International Workshop on Adaptivity, Personalization & the Semantic Web, workshop at Hypertext 2006, Odense, Denmark, Aug, 2006.

[14] L. Aroyo, P. Dolog, G.-J. Houben, M. Kravcik, A. Naeve, M. Nilsson, and F. Wild. Interoperability in Personalized Adaptive Learning. Educational Technology & Society, 9(2):14-18, 2006.

[15] Yahoo! Pipes. http://pipes.yahoo.com/pipes/

[16] Mircosoft Popfly. http://www.popfly.com/

[17] Google Mashup Editor. http://editor.googlemashups.com/

[18] R. Ghosh, and M. Dekhil. Mashups for Semantic User Profiles. In Proc. of the 17th International Conference on World Wide Web (WWW 2008), Beijing, China, Apr, 2008.

[19] R. Ghosh, and M. Dekhil. I, me and my phone: identity and personalization using mobile devices, HP Labs Technical Report HPL-2007-184, Nov, 2007.

[20] H. Guo. SOAP: Live Recommendations through Social Agents. In Fifth DELOS Workshop on Filtering and Collaborative Filtering, Budapest, 1997.