Towards understanding the needs of cognitive support for ontology mapping

Sean M. Falconer[1], Natalya F. Noy[2], and Margaret-Anne Storey[1]

[1] University of Victoria, Victoria BC V8W 2Y2, Canada
{seanf, mstorey}@uvic.ca
[2] Stanford University, Stanford, CA 94305, USA
noy@stanford.edu

Abstract. Researchers have developed a large number of ontology-mapping algorithms in recent years. However, ontology mapping is hardly a fully automated task and users must verify and fine-tune the mappings resulting from automated algorithms. Both academic and industry researchers have focused on the algorithms themselves and largely ignored the issue of cognitive support for users in the task of analyzing mappings proposed by the algorithms and creating new mappings. The lack of comprehensive user-oriented tools for ontology mapping (rather than just algorithms) hinders the adoption of the new technologie. In this paper, we analyze requirements for cognitive support for the ontology-mapping task. Recognizing that many researchers must focus on improving the algorithm performance itself (or only on providing better visualization), we have developed a plugin framework that enables developers to assemble a comprehensive ontology-mapping tool by plugging in various components. We provide a reference implementation of the complete framework. Thus, developers can plug in only the components they are interested in. For example, algorithm developers can plug in their algorithm and use the visualization components that we provide and the user-interface researchers can use the framework to experiment with various visualization paradigms for ontology mapping (and not worry about implementing the algorithms themselves). We also discuss specific cognitive aids for ontology mapping that we have developed and that are available as part of this framework.

1 Introduction

As ontologies become more commonplace and their number grows, so does their diversity and heterogeneity. Reconciling different ontologies and finding correspondences between their concepts is likely to be a problem for the foreseeable future. Thus, research on ontology mapping has become a prominent topic in the Semantic Web and ontology communities. There are mapping contests that compare the effectiveness of different algorithms [10], and researchers have proposed a standard mapping language [11]. As the results of the contests show, ontology mapping is far from being a fully automated task. We believe that in most cases manual intervention will be required to verify or fine-tune the mappings produced by the algorithms.

In general, a user interacting with an ontology-mapping tool, must examine the *candidate mappings* produced by the tool, indicate which ones are correct and which ones are not, and create additional mappings that the tool has missed. This process is

a difficult cognitive task. It requires understanding of both ontologies being mapped and how they relate to each other. Also, both the ontologies themselves and the number of candidate mappings that the tools produce can be very large. However, there has been very little research on how to provide cognitive support for ontology mappings. Researchers have focused on improving the performance of the algorithms themselves, largely ignoring the issue of *end-user tools* (with a few exceptions [12, 18, 24]).

Perhaps, one of the reasons this issue has not been addressed is the lack of understanding of the importance of cognitive support [23]. Cognitive support measures how well a tool supports a user's cognitive processes, and it results from the interplay between the system image and the user's needs [8]. As Walenstein states, "The first rule of tool design is to make it useful; making it usable is necessarily second, even though it is a close second . . . [A tool's] usefulness is ultimately dependent upon [its] utility relating to cognition: i.e. to thinking, reasoning, and creating. Assistance to such cognitive work can be called cognitive support." [23]

Cognitive-support research is still new to software and knowledge engineering. In software engineering, generally, most tools are built with some consideration of utility and usability. Only recently, researchers started using a more formal approach to address these issues. The situation is similar in the field of knowledge-engineering tools. In this paper, we bring cognitive-support considerations to the ontology-mapping tools, and outline a set of requirements for these tools.

We believe that in order for the ontology-mapping tools to reach beyond research labs, both the performance of automatic ontology-mapping algorithms and the quality of cognitive support in ontology-mapping tools must improve. Recognizing that in many cases, researchers must focus on one or the other of these tasks, we have developed a plugin framework that covers many of the sub-tasks of ontology mapping, from specifying algorithms for initial comparison to executing the mappings. This framework is part of the PROMPT ontology-management suite [18], itself a Protégé plugin.[3] We have developed a reference implementation for each of the steps, including a number of cognitive aids. Developers can plug in their own components and use plugins developed by others (including our team) in order to fill in the missing pieces to have a comprehensive end-user tool.

This paper makes the following contributions:

- We analyzed requirements for cognitive support in ontology mapping (Section 2).
- We developed the PROMPT plugin architecture for ontology-management that enables developers to assemble a comprehensive ontology-mapping tool with their own components as part of the tool (Section 4).
- We implemented a set visualization plugins to the PROMPT plugin architecture that provides cognitive support for users in the ontology-mapping task (Section 5.2)

2 Requirements for Cognitive Support in Ontology Alignment

The set of end-user tasks in an ontology-mapping tool that we identified and the corresponding requirements are based on the common problems we have experienced and

[3] `http://protege.stanford.edu`

witnessed. We assume that the user's tasks involve verifying and fine-tuning the mappings produced by the automatic component and creating the mappings that the automatic algorithm missed. The following is a preliminary list of tasks that must be supported during the mapping process. Some of the requirements below address visual aids that make the user's job easier, and others are tasks that help reduce the user's cognitive load.

Navigation of ontologies being mapped: provide full access to the source and target ontologies.

Incremental navigation: enable browsing of the ontologies being mapped with the terms in the current mapping as focal points. Incremental navigation restricts the focus to the terms and allows the user to visually verify that the two terms suggested in the mapping have similar context and similar neighbors.

Identification of "candidate-heavy" ontology regions: identify visually which sections of the ontologies have large numbers of candidate mappings. Users may often want to focus on the sections where many of the mappings are, since these are likely to be the sections of the two ontologies where most of the mapping takes place.

Browsable list of candidate mappings: provide easy navigation and filtering of the candidate mappings produced by the automatic step. There must be a way for the user to instruct the tool to focus on certain regions of the ontologies or categorize the candidate mappings they are verifying. Such support allows the user to focus on smaller tasks and reduce complexity by validating higher priority matches first.

Information about the reasons a mapping was suggested: provide the user with some indication of why the automatic algorithm suggested a particular mapping. This reason helps establish trust between the user and the algorithm. For example, state that the two ontology terms matched exactly, or were synonyms of each other.

Context for mapping terms: display where the terms being mapped are in the ontology. Easy access to this information is essential in enabling the user to verify candidate mappings. In particular, the neighborhood of a term (immediate parent and children in the `is_a` hierarchy) may be especially important.

Definitions for mapping terms: provide easy access to full definitions of the terms in the ontology. For example, the definition might include the properties of a class and restrictions on those properties. Like the neighborhood, the internal structure helps explain the meaning of the term.

Conflict resolution and inconsistency detection: indicate to the user if some of the mappings that he has created produce conflicts or are inconsistent. Conflicts can arise from a variety of situations, such as when two concepts are mapped, but some structural elements that are critical for their definition have not been mapped yet.

Ability to save the verification state: The verification process must support potential interruptions where the user must be able to save their current progress and restart from that point at a later time.

Verification of mappings through execution: enable users to "execute" the mappings, for example, by transforming instances from the source to the target ontology based on specified mapping or using queries that access those individuals but this time direct them to the newly mapped term One can view such a transformation as a *debugging* step in creating a complete mapping: the user can verify if the instances created in the target from the source instances are the ones that he expected.

Direct creation and manipulation of the mappings: enable users to add details to the verified mappings. For example, a user may specify that a value of one property must be changed in a specific way in order for the mapping to be correct. Also, users may want to add metadata to mappings and describe their reasons for creating the mapping. The users will also often add mappings that the tool has missed.

Navigation of verified and manually specified mappings: One result of the user interaction with a mapping tool is a set of verified mappings, additional mappings, and details about the mappings. Users must be able to navigate this information.

Progress feedback: inform the user about their current progress in the mapping. How much they have verified and how much is left to verify. Verifying mappings can be a lengthy process, but providing feedback about the users' progress enables them to see that they are moving in the correct direction.

3 Related Work

Cognitive support is about introducing artifacts in order to improve cognitive systems [23]. As Norman states [17], "The power of the unaided mind is highly overrated. Without external aids, memory, thought, and reasoning are all constrained." Although cognitive support can be addressed in a variety of ways, one popular approach is through *information visualization*. Information visualization leverages innate human abilities to perform spatial reasoning and make sense of relatively complex data using some form of graphical representation language [9]. Information visualization is often used to construct an advanced user interface to aid humans understand and navigate complex information spaces. In software engineering, this approach has been applied specifically to applications such as source code evolution [22] and algorithm animation [2].

Knowledge-engineering tools often use visualization to help users navigate ontologies. Usually, the problem these visualization tools are addressing is the comprehension and navigation of large information spaces. Different types of graph layouts are often used in order to display the ontology from different perspectives (e.g., see [21]). One of the goals of providing these various layouts is to help users view the same knowledge in different formats and potentially to validate and invalidate their mental models. As Richer and Clancy state, "...providing multiple views of the same knowledge or behavior can help a user understand a complex system." [19]

Navigation is also a relevant issue in ontology mapping, as users need to understand the structural context related to the match operations they must verify. However, in ontology mapping the focal point of the navigation is the terms involved in the mapping, not necessarily the entire information space.

3.1 Ontology-Mapping Tools and Their User Interfaces

Most user interfaces for mapping tools fall into one of three categories: graphical user interface, console-based, and finally web-based. Both the console-based and web-based tools follow similar approaches; the user supplies URIs for two ontologies, submits the input, and the tool processes the ontologies producing a list of potential matches. FOAM [7], MoA Shell [13], Chimaera [16], and the OWL Ontology Aligner [25] all

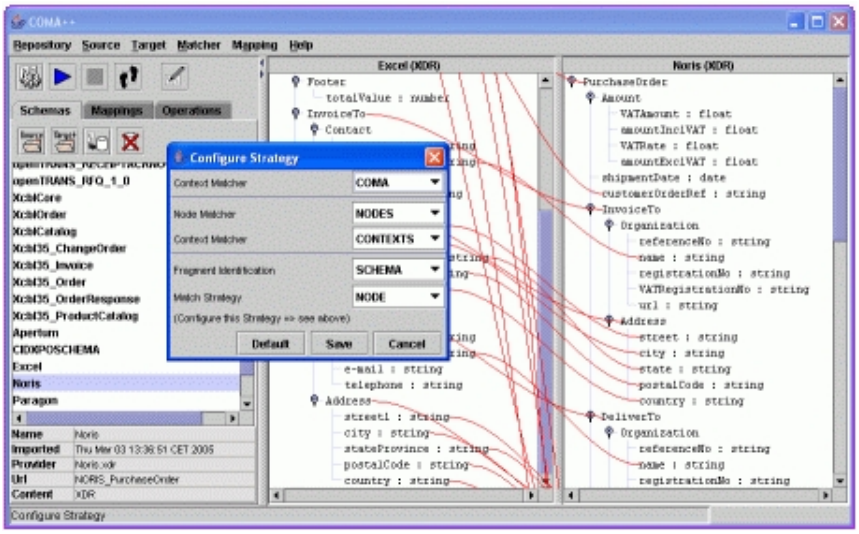

Fig. 1: Mapping two schemas in COMA++.

fall into one of these two categories. Some of these tools, such as FOAM, also support interactive modes where a user can verify matches as they are computed.

Clio [12] and COMA++ [5] are examples of tools that support graphical user interfaces. The number of visual paradigms that the tools use to display the mappings is quite limited, however. Clio was developed by IBM for generating mappings between relational and XML schemas. Clio can infer correspondences in the source and target schemas and it also allows users to draw correspondences between parts of the schemas. Once the correspondences have been generated and verified, Clio generates queries to drive the translation from the source schema to the target schema. COMA++ works similarly, although it also supports ontology mapping. COMA++ automatically generates mappings between the source and target schemas, and draws lines between matching terms. Users can also define their own term matches (see Fig. 1). Both tools draw mappings between the source and target schemas, which can be difficult to work with when there are a lot of mappings or the distances between mapped terms is large.

These tools provide a mechanism to allow the user to supply an initial set of matches. This mechanism may be adapted to store a partially verified mapping and to restart verification at that particular stage. Also, tools like Clio and COMA++ support in-tool navigation of ontologies. There's also recently been an effort by a small number of researchers towards investigating applying visualization techniques to ontology alignment. AlViz [15], a plugin for Protégé, applies multiple-views via a cluster graph visualization along with synchronized navigation within standard tree controls. Generally, there is a dearth of visual paradigms for ontology mapping. Until we have such end-user tools with good cognitive support, many of the mapping algorithms that researchers develop, are unlikely to leave the labs. Hence, we have developed a plugin framework that treats an ontology-mapping process as a standard sequence of steps (from initial comparison of ontologies to executing mappings) and enables developers to substitute any of the steps with their own tools.

3.2 Component Frameworks for Ontology Mapping

Several researchers have addressed the issue of deomposing an ontology-mapping process into a sequence of subtasks–a step necessary for introducing a plugin framework for ontology mapping [6, 14]. These subtasks include pre-processing of the source ontology, configuration of the mapping algorithm, analysis of the results, and iterative invocation of the mapping algorithm. In a sense, implementing a plugin framework for ontology mapping starts with identification of this set of tasks. However, in order to enable developers to substitute implementation of any of the tasks with their own, we also must define interfaces, and must provide extension points in the tools.

The work that is closest to our plugin framework for mapping is the IBM XML Mapping technology [20]. In this work, the authors developed a plugin framework for mapping data sources such as relational database schemas, UML models, and XML files to XML Schema. Their architecture distinguishes four core components, each of which has extension points for plugins: (1) user interface, with plugins for viewing different types of models; (2) mapping population, allowing developers to plugin different mapping algorithms; (3) mapping representation, enabling different forms of representing the mappings; and (4) code generation, providing runtime engines for executing the mappings. We take a similar approach in our work. However, we focus more on cognitive support for mapping, with user interface being a very prominent component. In addition, our work is applied to ontologies and not only to XML schemas.

4 PROMPT Plugin Architecture

PROMPT is a Protégé plugin that supports various tasks for managing multiple ontologies, including ontology mapping [18]. In its original form, PROMPT starts the ontology-mapping process by performing initial comparison of the source and target ontologies to be mapped, mainly based on lexical comparison of class names. After the initial comparison, PROMPT presents the user with a set of candidate mappings. A user can examine the mappings, create new mappings, and save the correct ones. As the user identifies a mapping as correct, PROMPT performs structural analysis of the neighborhood of the mapped concepts, suggesting new mappings based on the graph structure. PROMPT saves the mappings as instances in the *mapping ontology* [3]. After the user defines the mappings, he can run a *mapping interpreter* [4] to transform instances from the source to the target ontology based on the mapping.

When verifying candidate mappings, users can access the source and target ontologies in the Protégé interface; when a user selects a mapping to examine, the corresponding concepts are highlighted in the ontologies trees. A user can also navigate to the tab that displays the mapping ontology and its instances and edit the mappings there directly (see Fig. 2).

The PROMPT plugin framework allows developers to replace any of the components that we have just described with their own. The plugin framework works by providing Java interfaces for various types of plugins (comparison algorithm, visualization components, etc). A plugin developer chooses the interface they wish to implement, and then supplies the appropriate method bodies in order to perform the operations they wish to

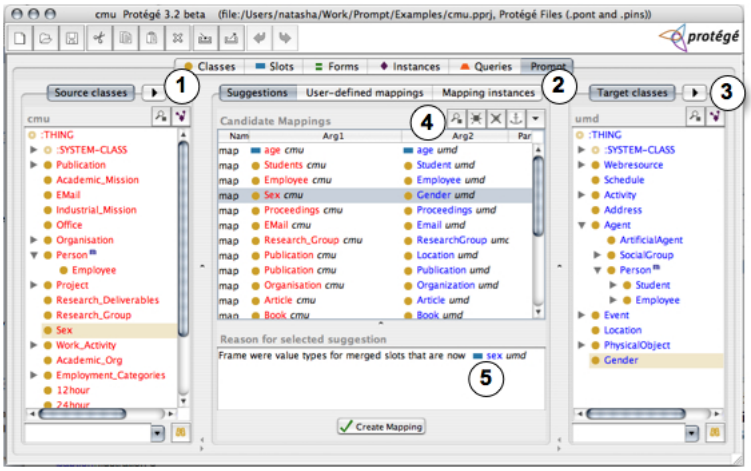

Fig. 2: The PROMPT user interface and the extension points in PROMPT's mapping component. The left column shows the source ontology; the middle column displays the mappings suggested by PROMPT and explanations of these suggestions. The right column displays the target ontology. There are tab extensions points for the source (1), mapping (2), and target (3) components. Area (4) shows the suggestion header button extension point. Algorithms can provide their own explanations for each candidate mapping (5).

execute. More specifically, we view the ontology-mapping process as a sequence of the following steps (Fig. 3):

Perform initial comparison of the ontologies: an algorithm compares two ontologies and produces a list of candidate mappings.

Present candidate mappings to the user enabling him to analyze the results. This step includes components for cognitive support (various visualizations of the source and target ontologies, options to filter content presented in the display, etc.) and interactive comparison algorithms that are invoked either explicitly by the user or as a result of mappings being verified.

Fine tune and save the mappings in a declarative mapping format.

Execute mappings to transform instances from source to target or to perform other operations.

In the current implementation, developers can replace components of any of the steps in this list, and our plan is to make all of the steps replaceable.

Figure 4 shows the PROMPT screen for configuring an algorithm for initial comparison. The user has chosen to run a FOAM algorithm at this stage. The integration of FOAM and PROMPT is available as part of PROMPT distribution [4]. A developer of an algorithm plugin can specify not only how to invoke the algorithm, but also how the configuration screen presented to the user should look like.

User-interface extension points exist throughout the PROMPT mapping interface. We currently support extensions that allow a developer to add new tabs to the source,

[4] At this time, the only algorithms integrated are FOAM and the original PROMPT algorithms.

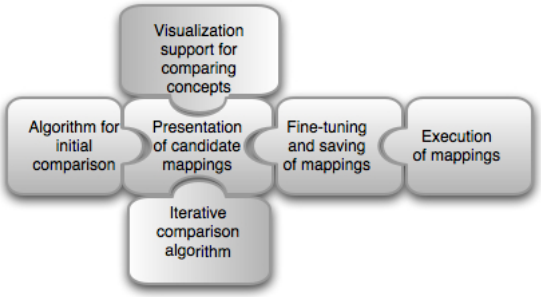

Fig. 3: Configurable steps in the PROMPT framework. Developers can replace any component in the figure with their own implementation. Cognitive aids can be applied at each step to ease cognitive load.

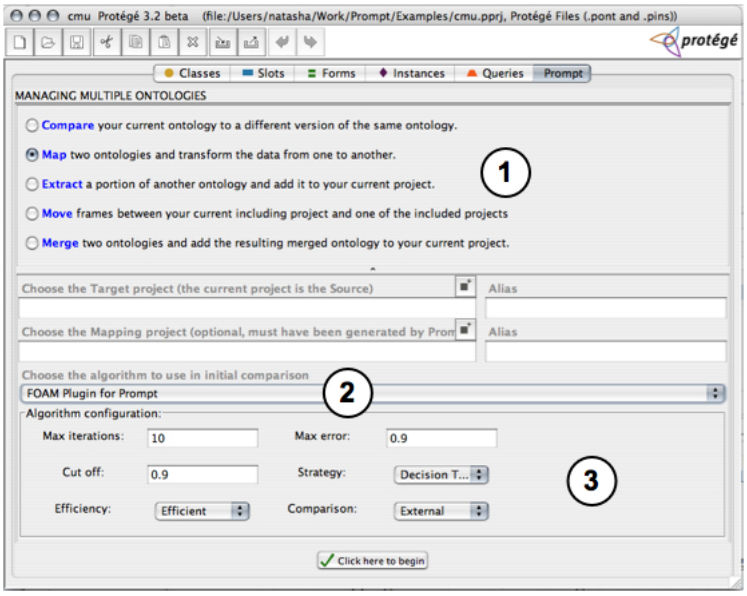

Fig. 4: Selecting an algorithm in PROMPT. The area marked (1) shows the options in PROMPT, here we have selected to map two ontologies. The area marked (2) displays the algorithm plugins available, here we have selected our custom built FOAM plugin. Finally, area (3) shows the algorithm configuration panel supplied by the FOAM plugin.

mapping, and target panels in the mapping component (see Fig. 2). Also, new header buttons can be added to the mapping suggestion list. For example, developers could use actions on these buttons to filter information (Section 5.2).

5 Cognitive Support in PROMPT

We examine issues of cognitive support of the core PROMPT plugin (before the addition of the interface plugins discussed in Section 5) as an example of questions that

arise when developing support for such a cognitively complex task as ontology mapping. PROMPT already addresses many of the cogntive support requirements that we discussed in section 2. However, PROMPT may not address all of these requirements so that users can make use of the features effectively. Meeting the requirements is not enough, we must also understand the usability of the implemented features.

5.1 Potential Problems in a Mapping User Interface

Although we have not yet carried out formal user studies to understand PROMPT's usability, we have identified some questions that such usability testing will reveal. For example, when a user selects a candidate mapping, PROMPT highlights the terms involved in the mappings in the source and target display. Thus, the user can see the context of the terms in the mapping. But this feature could potentially introduce new cognitive issues. For example, the selection of the terms in the ontologies is immediate, the ontology trees are expanded directly to the term that needs to be displayed, no animation of this process exists. Does jumping immediately to the term destroy the user's global context about where they are in the two ontologies? Does it interrupt the user's work flow? What if the user had used the ontology tree to browse to a particular location, but selecting a suggestion removed the user's selected focal point?

Another related issue is incremental navigation in PROMPT. Currently, the user can browse the source and target ontologies via a tree control in the mapping component. However, selecting a suggestion immediately switches the tree's focus. Also, this type of view can be difficult to use when viewing items deep in the hierarchy.

PROMPT loads all its generated mapping operations into a browsable suggestion list. With large ontologies, this list could be very large. There is currently no support for sorting, categorizing, or filtering the list. Abrams and colleagues [1] found that web browser users will not put more than 35 items in their favorite's list before resorting to categorizing links within hierarchies or stopping their use of favorites all together. Similar issues may need to be addressed in PROMPT. For example, what will users do when presented with a list of a thousand or even a hundred suggestions?

The final issue with PROMPT we wish to discuss is browsing the resulting mappings. Unlike Clio and COMA++, which draw lines between matching terms, PROMPT takes a different approach. Firstly, after the user confirms a mapping suggestion, the corresponding terms in both the source and target ontologies get a "mapped" icon associated with them indicating that the terms have already been mapped. Secondly, the occurance of the mapping event is recorded in a mapping ontology that is browsable by the user. Although the icon indicator certainly is less cluttered than the line drawing in Clio and COMA++, there is no explicit way for the user to visualize what the corresponding matched term is.

5.2 COGZ Interface Plugin for PROMPT

In order to address some of the missing requirements in PROMPT, we have developed COGZ—a user-interface plugin for the mapping component. COGZ attempts to provide user support for reducing the size and complexity of a mapping and to improve user interaction with establishing the term context and improving incremental navigation.

We addressed the complexity and size requirement by adding filters to the list of candidate mappings. There are several types of filters. First, users can filter candidate mappings based on the explanation provided by the algorithm that generated the candidate (see area (5) in Fig. 2). For example, the user can filter the list to inspect only exact term matches first, and then address more complex matches, like synonym or shared-hierarchy matches. There is also a filter that allows users to restrict mappings to classes from certain subtrees in the ontologies. Users can specify multiple subtrees in both the source and the target ontologies. This filter provides a powerful means for the user to address areas of the ontology they are most familiar with.

Finally, to address the context and navigation requirements, we added a *Neighborhood* tab to both the source and target ontology components. The neighborhood tab is synchronized with the browsing of candidate mappings: selecting a mapping displays the corresponding term's neighborhood. The neighborhood consists of the immediate parents and children of the term. The viewer supports incremental navigation by allowing the user to expand incrementally the neighborhood of any visible node. Also, the plugin provides six different layouts to allow the user to view the graph from a multitude of perspectives (see Fig. 5). The visualizations are provided by Jambalaya [21].

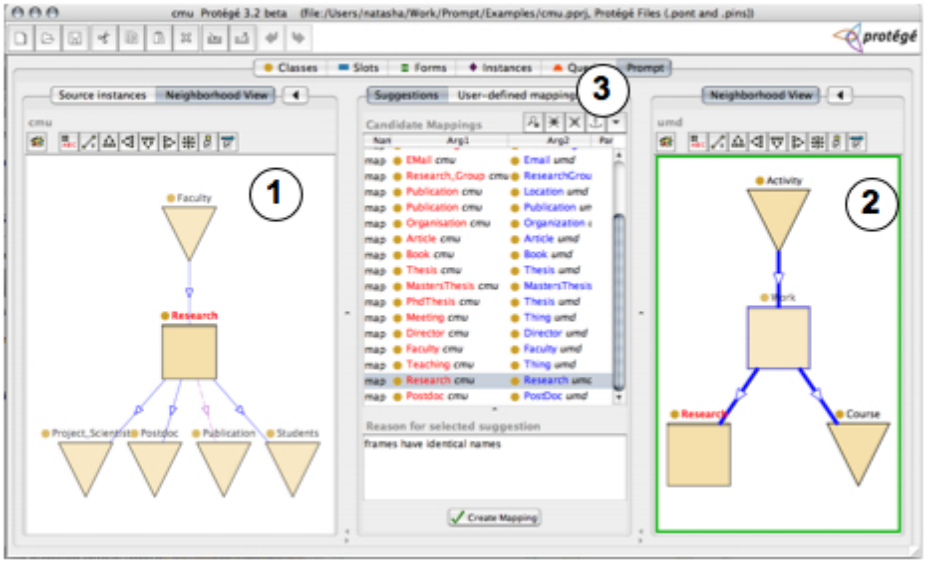

Fig. 5: PROMPT's mapping component with the interface plugin. Areas (1) and (2) show the neighborhoods of the source and target terms of the currently selected suggestion. Area (3) shows the location of the filter button. The neighborhood display makes it clear that the two classes have different meaning in the two ontologies even though their names are the same (*Research*).

6 Discussion and Future Work

We have discussed the need for cognitive support in ontology mapping tools. We proposed several requirements for satisfying this need, which were based on our own background work and experience. While these requirements are preliminary, we believe they represent a good initial description of the problems faced by users performing mappings. We believe it is very important to refine these requirements by carrying out studies and surveys in the knowledge engineering community. Specifically, we plan to evaluate the effectiveness of PROMPT through user studies and tool usage statistics. We also plan to enhance the visualization plugin in order to further address the requirements that PROMPT does not fullfill.

We also discussed the implementation of a plugin framework for PROMPT. The framework helps address two fundamental issues. Firstly, how can we satisfy the cognitive support requirements in one consistent environment, and secondly, how can we close the gap between mapping algorithm research and mapping users. By supporting user interface extension points in PROMPT, experts and developers in Human Computer Interaction can incorporate their ideas and tools to help decrease the cognitive load on end users of mapping tools. Similarly, the algorithm extension points also help the algorithm researcher. By using these extensions, researchers (or software developers) can easily incorporate their algorithms into PROMPT, allowing the research to be available under one consistent user interface. End users will benefit from having access to the best known algorithms, as well as the best cognitive support tools available.

In addition to developing better cognitive support for mappings, other research challenges remain. Our decomposition of the mapping process (Fig. 3) may not be general enough. It does not account fully for comparison algorithms that require an initial set of mapped terms as input. One can invoke such an algorithm at the iterative step, but more direct support for specifying the inputs precisely will likely be required. Similarly, PROMPT assumes a declarative representation of mappings (mainly as instances in an ontology). We would like to extend it to allow the use of an alignment API (e.g. [11]) and in-memory access to mappings. We envision that as developers begin to use the plugin framework, we will need to introduce other extension points of this type.

We plan to further enhance the plugin framework by adding more extension points for algorithms and interface components. PROMPT, FOAM and COGZ plugins are available as part of the full installation of Protégé 3.2beta.[5] Instructions for plugin developers and additional information are available on the PROMPT wiki site.[6]

Acknowledgements

This work was supported in part by the National Center for Biomedical Ontology, under roadmap-initiative grant U54 HG004028 from the National Institutes of Health.

[5] http://protege.stanford.edu/download/download.html
[6] http://protege.cim3.net/cgi-bin/wiki.pl?Prompt

References

1. D. Abrams, R. Baecker, and M. H. Chignell. Information archiving with bookmarks: Personal web space construction and organization. In *Human Factors in Computing Systems (CHI 98)*, pages 41–48. ACM Press, 1998.
2. M. D. Byrne, R. Catrambone, and J. T. Stasko. Evaluating animations as student aids in learning computer algorithms. *Computers & Education*, 33(4):253–278, 1999.
3. M. Crubézy and M. A. Musen. Ontologies in support of problem solving. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, pages 321–342. Sringer, 2003.
4. M. Crubézy, Z. Pincus, and M. Musen. Mediating knowledge between application components. In *Workshop on Semantic Integration at ISWC-2003*, Sanibel Island, FL, 2003.
5. H.-H. Do. *Schema Matching and Mapping-based Data Integration*. PhD thesis, Department of Computer Science, Universitt Leipzig, 2006.
6. M. Ehrig and Y. Sure. Ontology mapping - an integrated approach. In *1st European Semantic Web Symposium*, Heraklion, Greece, 2004. Springer, LNCS.
7. M. Ehrig and Y. Sure. FOAM–framework for ontology alignment and mapping; results of the ontology alignment initiative. In *Proceedings of the Workshop on Integrating Ontologies*, volume 156, pages 72–76, October 2005.
8. N. Ernst. Towards cognitive support in knowledge engineering: An adoption-centred customization framework for visual interfaces. Master's thesis, University of Victoria, 2004.
9. N. A. Ernst, M.-A. Storey, and P. Allen. Cognitive support for ontology modeling. *International Journal of Human-Computer Studies*, 62(5):553–577, 2005.
10. J. Euzenat. Eon ontology alignment contest. http://oaei.inrialpes.fr/2004/Contest/.
11. J. Euzenat. An api for ontology alignment. Technical report, 2006.
12. M. A. Hernandez, R. J. Miller, L. M. Haas, L. Yan, C. T. H. Ho, and X. Tian. Clio: A semi-automatic tool for schema mapping. In *SIGMOD Record*, 2001.
13. E. . T. R. Institute. Moa shell. http://mknows.etri.re.kr/moa/docs/moashell.html, 2003.
14. Y. Kalfoglou and M. Schorlemmer. IF-Map: an ontology mapping method based on information flow theory. *Journal on Data Semantics*, 1(1):98–127, Oct. 2003.
15. M. Lanzenberger and J. Sampson. Alviz - a tool for visual ontology alignment. In *IV '06: Proceedings of the conference on Information Visualization*, pages 430–440, Washington, DC, USA, 2006. IEEE Computer Society.
16. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. The chimaera ontology environment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 1123–1124. AAAI Press / The MIT Press, 2000.
17. D. A. Norman. *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*. Addison-Wesley, 1993.
18. N. F. Noy and M. A. Musen. The PROMPT suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.
19. M. Richer and W. J. Clancy. Guidon-watch: A graphic interface for viewing a knowledge-based system. *IEEE Computer Graphics and Applications*, 5(11):51–64, 1985.
20. M. Roth, M. Hernandez, P. Coulthard, L. Yan, L. Popa, and C. Salter. Xml mapping technology: Making connections in an xml-centric world. *IBM Systems Journal*, 45(2):389–409, 2006.
21. M.-A. Storey, N. F. Noy, M. Musen, C. Best, R. Fergerson, and N. Ernst. Jambalaya: an interactive environment for exploring ontologies. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 239–239, New York, NY, USA, 2002.
22. L. Voinea, A. Telea, and J. J. van Wijk. Cvsscan: visualization of code evolution. In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, pages 47–56, New York, NY, USA, 2005. ACM Press.
23. A. Walenstein. *Cognitive support in software engineering tools: A distributed cognition framework*. PhD thesis, Simon Fraser University, Vancouver, BC, 2002.
24. D. Wenke, S. Atanassov, D. Manov, M. Maier, and W. Sperling. D4.5.1 report on ontology mediation as service component. Technical report, EU-IST SEKT Deliverable, 2003.
25. A. V. Zhdanova. Owl ontology aligner. http://align.deri.org:8080/deri/align.jsp.