EASIER: An Approach to Automatically Generate Active Ontologies for Intelligent Assistants

Martin Blersch
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: blersch@kit.edu

and

Mathias Landhäußer
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: landhaeusser@kit.edu

ABSTRACT

Intelligent assistants are ubiquitous and will grow in importance. Apple's well-known assistant Siri uses Active Ontologies to process user input and to model the provided functionalities. Supporting new features requires extending the ontologies or even building new ones. The question is no longer "How to build an intelligent assistant?" but "How to do it efficiently?"

We propose EASIER, an approach to automate building and extending Active Ontologies. EASIER identifies new services automatically and classifies unseen service providers with a clustering-based approach. It proposes ontology elements for new service categories and service providers respectively to ease ontology building.

We evaluate EASIER with 292 form-based web services and two different clustering algorithms from Weka, DBScan and spectral clustering. DBScan achieves a F_1 score of 51% in a ten-fold cross validation but is outperformed by spectral clustering, which achieves a F_1 score of even 70%.

Keywords: Natural Language Understanding, Ontology Building, Clustering, Web Mining, Service Discovery.

1. INTRODUCTION

Intelligent assistants such as Siri are more popular than ever. In the beginning, they seemed to be toys solely designed to attract technophile customers. But as their capabilities and performance grew, more and more users use their intelligent assistants seriously to perform everyday tasks. But progress came with a price: The underlying logic, Active Ontologies (AOs) in the case of Siri [1], is hand-crafted for every single function. Natural language processing (NLP) – as well as natural language understanding (NLU) – is encoded within the AO's concepts. Tuning the NLU functionality is difficult.

Conceptually, there is one distinct AO for every service that an intelligent assistant offers. The ontology models the data necessary for the service and further optional details that the user can give. In addition, the concepts within the ontology contain code that is executed whenever the user interacts with the assistant. The code and its execution make the ontologies *active*. But AOs do not provide services by themselves. They only capture the user's intent and forward their commands to other software components to execute them. Because of that, they can be seen as user interfaces for a service oriented architecture (SOA). Adding a new service provider (SP) is easy, as long as it provides a functionality already modeled in an AO. If the SP provides something different, a new AO must be designed to model the mandatory (and optional) input data. As NLP is part of the AO, modeling the input data is not enough. One also has to encode the conditions and actions that process the natural language input in the newly created AO. In short, the implementation of new functionality is labor intensive.

We propose EASIER, a framework that simplifies integrating new functionality in Active Ontologies. EASIER defines a process for identifying new SPs for existing AOs and for creating new AOs for unseen services. Also, EASIER can make use of services offered as web (HTML) forms, as many services that target end-users do not provide standardized web services. Web forms are very popular because they are easy to use. Web users with average experience usually have less problems in formulating form-based queries compared to formulating queries in structured languages (e.g., SQL). The wide distribution of web forms makes them a perfect target for automated service integration. Other service types, such as XML-RPC and SOAP web services, can easily be integrated into EASIER using specialized communication interfaces.

This paper details the overall approach of EASIER and explains how we identify new services, add them to existing service categories, and how we identify new service categories automatically. The structure of this paper is as follows. Section 2 reviews related work. Then we explain how intelligent assistants are built using AOs and discuss the challenges. Sections 4 and 5 explain our approach and present an evaluation of the first steps, namely identifying services, building service categories, and deriving AOs from category descriptions. Then we discuss future work and conclude the paper.

2. RELATED WORK

EASIER needs to automatically derive ontologies from service descriptions. It builds upon results from different research areas. The first part of this section presents approaches for querying databases and forms with natural language. Then we present approaches for extracting ontologies from (web) forms. The third part presents different approaches for clustering (web) services.

Querying Databases

Interacting with web forms is similar to querying databases. Research in natural language interfaces to database systems (NLIDB) has a long history. Androutsopoulos et al. [2] give an introduction to NLIDB and present some of the linguistic problems they encountered. Pazos R. et al. [3] review the state of the art.

Meng [4] presents an NL interface for retrieving information from web forms. Words and phrases often have multiple meanings; therefore, it can be hard to determine which form field to fill the given information in. To solve this problem, he uses n-gram statistics to examine the words' contexts. If his system does not reach a proper conclusion, it asks the user. EASIER faces a similar problem when analyzing a user's input.

Álvarez et al. present DeepBot, a hidden web crawler, that interacts with web forms [5]. DeepBot automatically identifies forms and learns to execute queries on them. It needs a set of form attributes (i.e. field names) and a set of queries (i.e. attribute-value pairs). Also it calculates a probability for a given form that indicates how relevant the form is for a specific domain.

WISE-Integrator creates a single interface for several related web forms [6]. It extracts labels and controls from the forms including their visual representation (i.e. the HTML layout), meta data, and semantic information derived from the labels. Then it uses a two-step clustering technique to pair identical elements of different web forms. WISE-Integrator also maps form-specific field names to global field names that are used in the integrated form.

Ontology Extraction

An et al. proposed an approach to map web forms to ontologies [7]. They use machine learning to discover the mappings but the ontologies must exist beforehand. EASIER must first create the ontologies but extending the set of service providers for a given ontology yields a similar problem.

OntoBuilder extracts ontologies from web forms and merges ontologies from similar forms. Automatic ontology matching supports merging ontologies. OntoBuilder considers the concepts' data types, their value constraints, the ordering within forms, and syntactic features (e.g. form labels). Fields with similar meanings are also identified using the forms' layouts [8].

Similar to ontology merging is ontology matching. It deals with finding semantically related entities in different ontologies. EASIER matches equivalent fields from different forms during clustering. Otero-Cerdeira et al. provide a literature review of the area of ontology matching [9] and Shvaiko provides an extensive list of publications [10].

Berlanga et al. present a method for semi-automatically building ontologies from forms [11]. Their tool FAETON extracts a logical model from a form's components, i.e. its labels and text input fields, their visual and geometric features (e.g. font size or color), and the component's type information. Domain ontologies and thesauri are used to annotate textual sections and control labels. The logical model and the annotated form elements are combined to build the ontology.

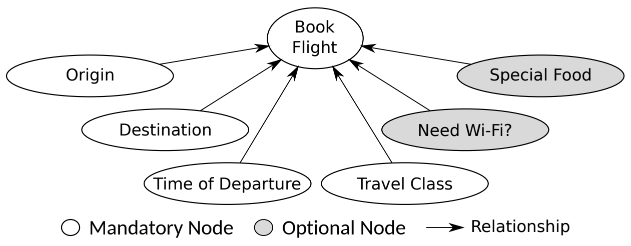


Fig. 1. An ontology modeling the input data for the flight booking domain. Optional information is not needed to book a flight.

Service Clustering

EASIER identifies service categories and new SPs for these categories by clustering. The following paragraphs present different approaches to clustering and give examples of their applications on web services and business processes.

Liang et al. take a two-tier clustering approach with k-means and bisecting k-means to cluster web services into categories [12]. They derive features for clustering from the WSDL descriptions and other meta data of the web services and take the categories from the UNSPSC taxonomy.

Reddy and Damodaram semantically cluster services of a SOA [13]. They use DBScan, a density-based clustering algorithm, to cluster the services based on the service descriptions. They pre-process the descriptions to remove stop words, to perform stemming, and to resolve synonyms; too general words are removed as well. The clusters are then used to semantically search for services instead of a keyword-based search.

Zhang et al. demonstrate how one can use spectral clustering to detect similar functionality in service-oriented architectures [14]. They build a call graph from interaction log files and derive a similarity matrix from the graph. Then a principal component analysis (PCA) is used to reduce the dimensionality of the matrix. Finally, k-means is used to build clusters. Reducing the dimensionality is necessary in our approach as well but the PCA creates new dimensions as linear combinations of existing ones, which can be hard to interpret. Furthermore, EASIER cannot use a PCA because the services' components are used in the AO for extracting *single* parameters from natural language input.

Jung et al. use hierarchical clustering to identify similar business processes [15]. A business process contains a number of activities, which are encoded as vectors. Then they compute the pairwise similarity between the activity vectors and use a weighted sum to determine the similarity of two processes. The clustering iteratively merges the two most similar clusters starting from single-element clusters until all elements are merged. With hierarchical clustering, one can choose the number of clusters a-posteriori.

3. System Overview

This section explains AOs in detail and provides an overview of the EASIER Active Server architecture.

Active Ontologies by Example

Guzzoni introduced AOs for natural language understanding [16], [17]. They proved to be especially useful for implementing intelligent assistants [1]. AOs model domain knowledge and combine that knowledge with an execution environment. Real world entities are represented as concepts, which can be connected by relationships. Information is stored as facts (i.e. logical predicates) in a global fact store. Figure 1 shows an example AO for the flight booking domain. The ontology models the information needed to make a booking: as shown, a place of origin, a destination, a departure time, and a travel class are mandatory for a booking. Yet, deciding whether one wants Wi-Fi or special food is optional.

AOs process natural language bottom-up: Incoming utterances are added to the fact store and processed by the leaf nodes. Every concept has a set of rules that are automatically evaluated when information is added to the fact store. An evaluation engine regularly checks the fact store for newly inserted or updated facts. If the facts have changed since the previous check, a new evaluation cycle begins. In an evaluation cycle, all rules are checked. The rules consist of one or more conditions and an action; if the conditions are satisfied, the respective node fires and the actions are executed. Unification (FOL) is used to identify facts, which match the conditions. Conditions of the leaf nodes' rules check the incoming utterances for specific information; e.g., the "destination" node seeks for airports or city names. The action part of the rule then forwards the acquired information along the relationships using communication pipes. Thus, relationships do not only model the structure of the domain knowledge but also the data flow in the AO. A leaf can fire multiple times in a single evaluation cycle if the relationship permits multiple messages. Also, actions can add facts to the fact store.

Inner nodes receive the messages from their children and then decide to send new messages to their parents. If the root node receives messages from all of its mandatory children, the desired task can be executed (i.e. calling an external service, creating new facts, etc.). When information is missing (e.g., a parent node receives messages from all but one of its mandatory children), the node can decide to ask the user.

As the natural language input can be generated by an automatic speech recognizer, the information does not need to be accurate. Therefore the messages from one node to its parents can contain additional information such as a confidence; the parent node then has to decide whether the confidence is high enough or whether the message should be discarded.

EASIER Active Server Architecture

The EASIER Active Server is a runtime environment for AOs and consists of the following components (cf. Figure 2): It has one *evaluation engine* and one *fact store*. Therefore, all AOs on one server have access to the same information.

Every task is modeled in one *task AO*[1]. The task AOs only gather the needed information and use *service providers* to fulfill the tasks. To decouple task AOs and SPs, the task AOs only need to know the service category. A special AO, the *broker*, forwards the requests to one or more service providers. A service category can include multiple service providers with the same functionality, e.g. two flight booking services. Also, connection handling, request submission and result processing is transparently handled by the broker. It uses the server's *communication interface* to contact the services.

If a dialog with the user is necessary, the task AOs rely on an other special purpose AO, the *dialog manager* (DM in Figure 2). The dialog manager uses the *dialog interface* to interact with the user. Task AOs do not need to know how to handle user interaction. To trigger user interaction, task AOs simply insert

[1] For the sake of simplicity, we assume that every task is modeled in a different task AO. Merging two similar task AOs is indeed possible and can be beneficial when the similar information is needed.

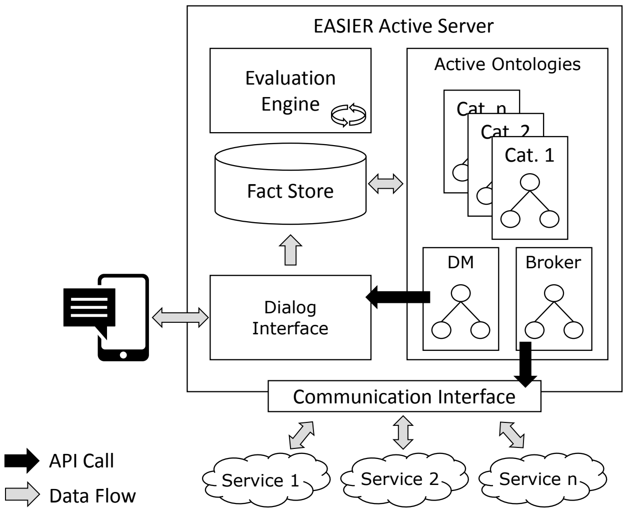


Fig. 2. The EASIER Active Server architecture, inspired by [17]. AOs use facts to communicate with each other. Special AOs, the dialog manager and the broker respectively, handle interaction with the user and service providers.

facts into the fact store to indicate that mandatory or optional information is missing or could be specified respectively.

Active Ontology Building Blocks

AOs contain leaf nodes and non-terminal nodes. Leaf nodes process input and send the extracted information to their parents. They can contain any user-defined rule but there are special leaves with the following predefined functionalities:

- *Vocabulary list* leaves compare incoming tokens with a user-defined list of words.
- *Prefix* and *postfix* leaves consider several tokens following/preceding one or more keywords.
- *Regular expression* leaves use a regular expression to identify well-formed tokens, e.g. ZIP codes or email addresses.
- *Specialized* leaves or groups, e.g. *date* and *time* leaves to identify exact dates and to determine relative dates such as "tomorrow" or "now".

Inner nodes can as well contain any user-defined rule but usually one of the following nodes is used:

- *Gather* nodes aggregate incoming information and report to their parents.
- *Select* nodes pick one of their child nodes' messages and forward it to their parents. Usually, they select the message with the highest confidence.

Usually the root node of an AO is a gather node that collects all information from its child nodes and triggers a command.

4. APPROACH

Today, intelligent assistants use pre-selected web services to fulfill user queries. However, a considerable amount of information is only accessible through form interfaces. Harnessing web forms as information resources for intelligent assistants would increase usability.

Manual construction of AOs is very time consuming. A new AO must be created every time when one extends the system with a new service category. This is currently done manually and is labor-intensive. At the moment, automatic connection of

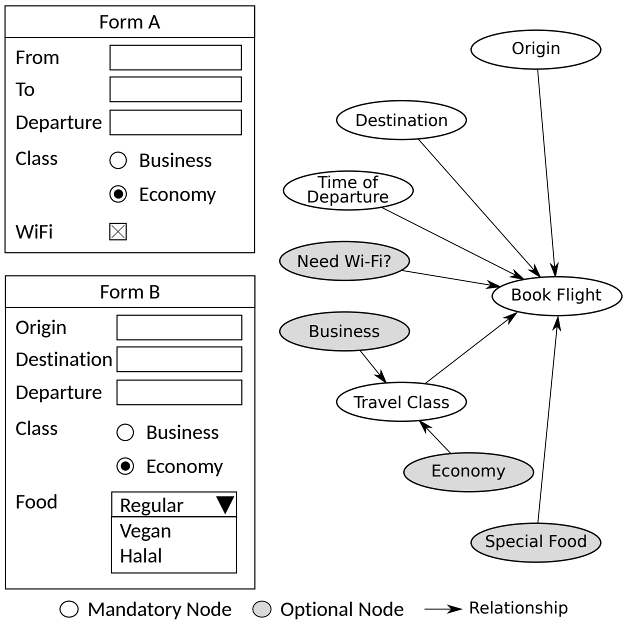


Fig. 3. Mapping forms to Active Ontologies. Commons fields are merged in the ontology; optional nodes represent form-specific fields.

HTML Form Element	Type	Sources for input validation
Radio button	V	buttons' values
Check box group	V*	check boxes' values
Select (single)	V	Options
Multi-Select	V*	Options
Color picker	V	Colors
Date or time picker	S	Date/time detection group
Input (with pattern)	R	RegEx from pattern attribute

TABLE I
DERIVATION RULES FOR AO LEAVES FROM FORM FIELDS (EXCERPT). TYPES: (V)OCABULARY LIST LEAF, (S)PECIALIZED NODE GROUP, (R)EGULAR EXPRESSION LEAF. (*) MARKS MULTI-VALUED INPUTS.

(new) Internet services is not possible. One problem is to map natural language to a form's fields. Field values can refer to several fields. Another problem is determining if the input is valid. Every field expects a certain kind of information. For many HTML input fields the expectation is not given explicitly. HTML5 offers new field types and provides validation support. E.g., for number fields, valid values and ranges can be specified by using the "number" attribute. Regular expressions can be defined to check the input fields' values against using the "pattern" attribute. Fields can be marked as required.

We propose to automate the creation of AOs. Our approach comprises the following steps: 1) We identify form-based services. We crawl the web for HTML forms and extract them with meta information. 2) We cluster the identified web forms to mine service categories using spectral clustering [18]. We considered DBScan [19] as well, but spectral clustering performs much better (cf. Figure 5). The clustering algorithms use a combination of the form elements' types, the frequency of their occurrence and their semantics to create features. The semantics can be determined by analyzing the form elements' labels. Some form elements can be specified in more detail by attributes like "name", "value", "title" or "placeholder". The attributes contain the elements' semantics and are used for matching fields from different forms with the same meaning by the clustering algorithm. We use WordNet [20] to determine the synonyms for the values to facilitate matching. 3) We automatically construct AOs for each service category from features that distinguish clusters. Each feature needed for clustering becomes one leaf node.

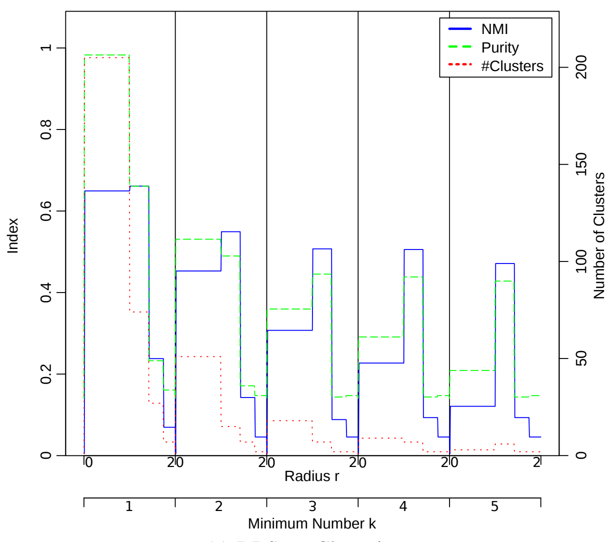
We derived a plan for constructing task AOs from the identified features. One task AO is created for each service category. Common features shared by all forms of a service category are modeled as mandatory nodes. All other features are optional (see Figure 1). To make the created ontology active, it is necessary to determine which node type to use for each node. The selection of the suitable node type depends on the form elements. Each node type must be provided with a list of valid values or pre- and/or postfixes, etc. Some form elements provide information about valid values or data types. E.g., a select element contains a list of valid options. EASIER uses this list to generate a vocabulary list leaf with the provided options. The "type" attribute of input elements indicates that the input must be in a certain format, e.g. a "date", "tel", etc. EASIER exploits the type to specify the input verification rules in the generated leaves, the regular expression pattern is extracted as well. Table I summarizes the derivation rules for AO leaves from HTML form fields. If no input validation hints are given, the information must be provided by the AO developer.

After creating the task AO representing one service category, it is registered with the EASIER system. For registering a service category, the following information must be provided: the unique name of the new service category (manually entered), all mandatory and optional parameters representing the nodes, and the parameter types e.g. "street", "telephone number", etc. SPs are registered separately using the name of the corresponding service category, a unique name for service identification and distinction, and information how to call the service. For form-based services, the URL (for single form) or URLs to all forms (for multi-stepped forms) are stored. For invoking the individual SPs with the correct parameter names, a mapping of all parameters to the category's AO must be provided. Such mappings are needed for each mandatory node of the task AO. Figure 3 shows the forms of two SPs and the corresponding task AO. Form A contains a field labeled "From" for filling in a flight's origin (i.e. an airport). The name of the corresponding task AO's node is "Origin". In this case, a mapping *A.From→Origin* is necessary and stored during the service registration of form A. Information from optional nodes can be used to further refine the desired service call.

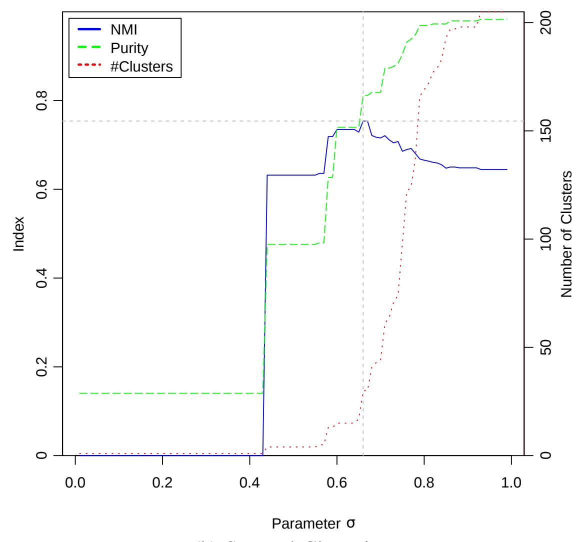
The broker uses this mapping information to select and invoke the individual SPs of the respective service category (as provided by the task AO) with the correct parameters. Three different invocation modes are supported by the broker: First, SPs can be invoked sequentially until the first SP returns a result which can be forwarded to the user. SPs can be invoked in parallel and one can tell the broker to wait for all SPs to respond or to return with the first received result: When using parallel-single mode, all SPs are invoked concurrently but only the first answer to a request is returned to the user. In parallel-all mode, the results of all SPs are collected, aggregated, and then returned to the user.

5. EVALUATION

We started a web crawler with seven URLs, such as Wikipedia's list of airlines, and assembled a set of 292 different web forms.



(a) DBScan Clustering



(b) Spectral Clustering

Fig. 4. In depth analysis of the clustering algorithms.

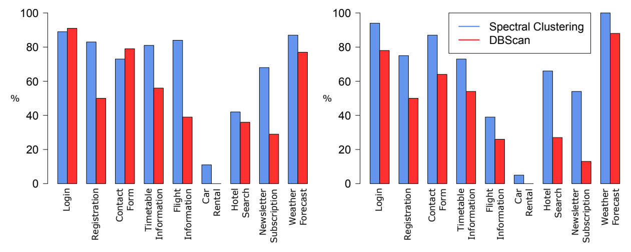
We labeled them manually with their service categories. We evaluate two classifiers using Weka's [21] implementation of DBScan and spectral clustering respectively. We perform a ten fold cross-validation with Weka to determine precision and recall of the trained classifiers and use 90% for training and 10% for testing.

Before that, we report an in depth evaluation of both clustering algorithms as they must be parameterized. The quality of clustering can be measured with *purity*. Its value ranges from 0 to 1, with 1 as optimal value. Purity determines the fraction of true positives in a cluster aggregated for all produced clusters. Perfect purity can be easily achieved, e.g. if every service gets its own cluster. To find a trade-off between the number of clusters and purity, one can measure the *normalized mutual information (NMI)*. It measures the purity of clusters but penalizes overfitting. NMI ranges from 0 to 1; it is 1, if every class is predicted perfectly by the classifier and every class forms one cluster.

Figure 4a shows the metrics for DBScan with different configurations. DBScan must be given minimum number of elements in a cluster $k \in \{0, \ldots, 5\}$ and a radius $r \in (0, 2]$. As one can see, purity and NMI decrease with increasing k for any given radius. DBScan achieves the best results for $r = k = 1$. The number of clusters with this configuration is quite high, 74, but the number drops to 15 for $k = 2$, which is too low. The best NMI is 0.66 for $r = k = 1$.

Figure 4b shows the metrics for the spectral clustering algorithm. Spectral clustering has only one parameter, σ, that is the similarity threshold used for edge pruning. It performs best with $\sigma = 0.66$ which results in 30 clusters with 81% purity. Spectral clustering outperforms DBScan with respect to NMI by ten percentage points on average.

Figure 5 shows precision and recall of both classifiers for nine service categories. DBScan achieves a precision of 55% and a recall of 47% on average. It performs best on login, contact, and weather information forms. Car rentals and newsletter subscriptions are similar to flight information and contact forms respectively. DBScan does not classify such forms properly, rendering the results for these categories unusable. Spectral clustering outperforms the precision of DBScan in all but



(a) Precision (b) Recall

Fig. 5. Comparison of DBScan and spectral clustering for the main service categories.

two categories, login and contact form; recall is better in all categories. On average, precision is 69% and recall is 70%.

6. Conclusion and Future Work

We presented Easier, a framework to build AOs from web forms automatically. Our preliminary results are promising but leave room for improvement.

Easier automatically identifies form-based services and classifies them using clustering techniques. For each service category, one AO is created using the derivation rules listed in Table I. The AOs are used for processing natural language and invoking the corresponding SPs. SPs and service categories are registered with the Easier framework. Additional form-based services can easily be classified in one of the existing categories and, after registration, invoked by the Easier Active Server.

Clustering is flexible and we evaluated DBScan and spectral clustering. The clustering algorithms achieve F_1 scores of 51% and 70% respectively but could be further improved. Our benchmark contains 292 web forms and will be further expanded. Future work will investigate if more training data improves precision and recall significantly or if other clustering techniques or an entirely different approach are needed.

Both algorithms lack precision and recall if service categories are too similar (e.g. car rental and flight booking). An intermediate approach could use gazetteer lists mapping company names to service categories. Also, classifying such services could benefit from meta data extracted from the corresponding web page (e.g. headings on the page) and its context (e.g. the entire Internet presence of a company).

EASIER automatically proposes ontologies for forms or ontology fragments for form elements that provide information about the expected data type and range. For unrestricted input fields, no ontology nodes can be proposed though. We want to decrease the manual effort further by drawing from additional information resources such as an upper ontology.

An active ontology editor could provide support in semi-automatically creating an AO. The tool could propose appropriate AO building blocks for form elements where EASIER cannot automatically select a mapping. Then the user could be prompted to select a suitable AO building block and to enter the form element's expected data type, range, or list of possible values.

At the moment, ontology generation does not regard the form layout. The order of fields or visual grouping of fields could provide further semantic information. Future work will also investigate how to obtain and use that information.

From an engineering point of view, there is work left to do. The EASIER Active Server supports off-line updates only, i.e. new services (and service categories) cannot be registered at runtime. We plan to expose the API for service category creation and service provider registration as a web service.

Before handing over the results to the user, they must be consolidated and cleaned up. Automating this process is still an open issue which requires further research.

Harnessing semantic web technologies should improve service identification, AO generation, and result extraction. Unfortunately, commercial web sites hardly provide semantic annotations.

REFERENCES

[1] J. R. Bellegarda, "Spoken Language Understanding for Natural Interaction: The Siri Experience," in *Natural Interaction with Robots, Knowbots and Smartphones*, J. Mariani, S. Rosset, M. Garnier-Rizet, and L. Devillers, Eds. Springer New York, 2014, pp. 3–14. [Online]. Available: https://dx.doi.org/10.1007/978-1-4614-8280-2_1

[2] I. Androutsopoulos, G. Ritchie, and P. Thanisch, "Natural language interfaces to databases – an introduction," *Natural Language Engineering*, vol. 1, no. 01, pp. 29–81, Mar. 1995. [Online]. Available: http://journals.cambridge.org/article_S135132490000005X

[3] R. A. Pazos R., J. J. González B., M. A. Aguirre L., J. A. Martinez F., and H. J. Fraire H., "Natural Language Interfaces to Databases: An Analysis of the State of the Art," in *Recent Advances on Hybrid Intelligent Systems*, ser. Studies in Computational Intelligence, O. Castillo, P. Melin, and J. Kacprzyk, Eds. Springer Berlin Heidelberg, 2013, no. 451, pp. 463–480. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-33021-6_36

[4] F. Meng, "A Natural Language Interface for Information Retrieval from Forms on the World Wide Web." Association for Information Systems, 1999, pp. 540–545. [Online]. Available: http://dl.acm.org/citation.cfm?id=352925.352991

[5] M. Álvarez, J. Raposo, A. Pan, F. Cacheda, F. Bellas, and V. Carneiro, "Crawling the content hidden behind web forms," *Computational Science and Its Applications–ICCSA 2007*, pp. 322–333, 2007. [Online]. Available: http://www.springerlink.com/index/1503244437564357.pdf

[6] H. He, W. Meng, C. Yu, and Z. Wu, "WISE-Integrator: A System for Extracting and Integrating Complex Web Search Interfaces of the Deep Web," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05. Trondheim, Norway: VLDB Endowment, 2005, pp. 1314–1317. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083592.1083761

[7] Y. An, X. Hu, and I.-Y. Song, "Learning to discover complex mappings from web forms to ontologies," 2012, pp. 1253–1262.

[8] A. Gal, G. Modica, H. Jamil, and A. Eyal, "Automatic ontology matching using application semantics," *AI magazine*, vol. 26, no. 1, p. 21, 2005. [Online]. Available: http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1796

[9] L. Otero-Cerdeira, F. J. Rodríguez-Martínez, and A. Gómez-Rodríguez, "Ontology matching: A literature review," *Expert Systems with Applications*, vol. 42, no. 2, pp. 949–971, Feb. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417414005144

[10] I. Pavel Shvaiko, TasLab, Informatica Trentina, "Publications: Ontology Matching," 2015. [Online]. Available: http://www.ontologymatching.org/publications.html

[11] R. Berlanga, E. Jimenez-Ruiz, V. Nebot, and I. Sanz, "FAETON: Form analysis and extraction tool for ontology construction," *International Journal of Computer Applications in Technology*, vol. 39, no. 4, pp. 224–233, 2010.

[12] Q. Liang, P. Li, P. Hung, and X. Wu, "Clustering Web Services for Automatic Categorization," in *IEEE International Conference on Services Computing, 2009. SCC '09*, Sep. 2009, pp. 380–387.

[13] P. Reddy and A. Damodaram, "Web services discovery based on semantic similarity clustering," in *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, Sep. 2012, pp. 1–7.

[14] X. Zhang, Y. Yin, M. Zhang, and B. Zhang, "Web Service Community Discovery Based on Spectrum Clustering," in *International Conference on Computational Intelligence and Security, 2009. CIS '09*, vol. 2, Dec. 2009, pp. 187–191.

[15] J.-Y. Jung, J. Bae, and L. Liu, "Hierarchical Business Process Clustering," in *IEEE International Conference on Services Computing, 2008. SCC '08*, vol. 2, Jul. 2008, pp. 613–616.

[16] D. Guzzoni, C. Baur, and A. Cheyer, "Active: A Unified Platform for Building Intelligent Web Interaction Assistants," in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology - Workshops, Hong Kong, China, 18-22 December 2006*. IEEE Computer Society, Dec. 2006, pp. 417–420.

[17] ——, "Modeling Human-Agent Interaction with Active Ontologies," in *nteraction Challenges for Intelligent Assistants, Papers from the 2007 AAAI Spring Symposium, Technical Report SS-07-04, Stanford, California, USA, March 26-28, 2007*. Stanford, California, USA: AAAI, Mar. 2007, pp. 52–59. [Online]. Available: http://www.aaai.org/Library/Symposia/Spring/2007/ss07-04-009.php

[18] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Aug. 2007. [Online]. Available: http://link.springer.com/article/10.1007/s11222-007-9033-z

[19] M. Ester, H.-p. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proceedings ot 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*. AAAI Press, 1996, pp. 226–231.

[20] C. Fellbaum, Ed., *WordNet: An Electronic Lexical Database*, ser. Language, speech and communication. Cambridge, Mass. [u.a.]: MIT Press, 1998.

[21] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278