Entity-Centric Search For Enterprise Services

Marcus Roy[1,2,3], Ingo Weber[2,3], and Boualem Benatallah[3]

[1] SAP Research, Sydney, Australia
[2] NICTA, Sydney, Australia*
[3] School of Computer Science and Engineering, Sydney, Australia
{m.roy,ingo.weber,boualem}@cse.unsw.edu.au

Abstract. The consumption of APIs, such as Enterprise Services (ESs) in an enterprise Service-Oriented Architecture (eSOA), has largely been a task for experienced developers. With the rapidly growing number of such (Web)APIs, users with little or no experience in a given API face the problem of finding relevant API operations – e.g., mashups developers. However, building an effective search has been a challenge: Information Retrieval (IR) methods struggle with the brevity of text in API descriptions, whereas semantic search technologies require domain ontologies and formal queries. Motivated by the search behavior of users, we propose an iterative keyword search based on entities. The entities are part of a knowledge base, whose content stems from model-driven engineering. We implemented our approach and conducted a user study showing significant improvements in search effectiveness.

1 Introduction

In many enterprise-level efforts of application development or integration, the search and use of APIs has traditionally been a task performed by internal and experienced developers. However, recently there has been a significant increase in providing publicly available APIs, particularly on the Web[1]. As a consequence, the level of experience a user can have with any given API decreases on average. Thus, finding the desired functionality within an API becomes more challenging. Examples of users who are typically inexperienced in a given API include Mashup developers wanting to create new composite applications; and consultants developing business scenarios on the basis of existing functionality. Hence we expect users to neither be experts in the API-related domain nor to know a specific query language to express their information need. We therefore believe it is crucial to provide such users with a highly effective, ad-hoc keyword search over API operations. In this paper, we consider a specific class of Web APIs, namely Enterprise Services (ESs): enterprise-class Web services [5], as are common in an eSOA. ESs are usually advertised through specific repositories, e.g. SAP's Enterprise Service Workplace and Registry (ESW) [18] or IBM's WebSphere Service Registry and Repository (WSRR[2]), which can easily contain thousands of services [16].

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

[1] http://blog.programmableweb.com/2012/11/26/8000-apis-rise-of-the-enterprise, accessed 12/12

[2] http://www-01.ibm.com/software/integration/wsrr, accessed 08/12.

To understand how users formulate free text queries, we investigated search logs from SAP's ESW. An initial analysis hereby revealed some common search patterns, i.e. users often started their search with a short query text using key business-related entities, e.g., "employee" and actions performed on these entities, e.g. "find employee". Motivated by the observations that users articulate their search needs using a small number of keywords [19] representing business entities or actions, and that they exhibit browsing-like behavior [7], we aim at providing an iterative keyword search over entities linked to ESs.

In this work, we tackle the difficult challenge to support keyword search over ES repositories: linking keywords used by users to concepts used by the service repository infrastructure to index and represent services, for which textual documentation is not always available. The proposed search technique relies on knowledge from model-driven engineering, including business entities consumed or generated by services and service operation patterns (e.g., create and read on business entities). In the following, we refer to the superset of business entities and action expressions as *entities*. In previous work, we presented an approach to automatically extract such entities [16] and learn naming conventions [17] from ES operation names, referred to as signatures. Using service design knowledge as an index over an ESs repository, we propose a ranking based on four different ranking measures related to entities. We implemented and evaluated the proposed keyword search and compared it to a state of the art IR-based search used at SAP, with significantly better results in terms of precision and recall[3]. In summary, the contributions are as follows: (i) an iterative search using entities extracted from service design knowledge, and (ii) an entity ranking using four different ranking measures related to entities.

2 Representing Service Design Knowledge

This section briefly revisits a formal representation of service design knowledge; for more details, we refer to [16]. To summarize the abstract representation of service design principles, we refer to a specific example of service design used in SAP, largely consisting (i) a business meta-data model, (ii) service design patterns and (iii) naming conventions as described hereinafter.

First, the business meta-data model generally defines a model of business entities, e.g. '`Sales Order`', '`Customer`' etc., used by both business and development departments. We describe this model using the MOF[4] layers from model-driven engineering. The M2-Model refers to metadata objects (and their relationship), e.g., '`Business Object`' (BO), which describes corresponding data objects (and their relationship) in the M1-Model, e.g., '`Sales Order`' as instances of `BO`.

Second, service design patterns are used during service development to describe the management of business entities and the behavior of respective ESs. Similar to the business meta-data model, models of service design patterns can be defined on M2-level to describe specific service design patterns on M1-level, e.g. '`Change`' as an instance of the '`Operation Pattern`' (OP).

[3] A detailed description of the evaluation experiment and result can be found in [15]

[4] Meta-Object-Facility (MOF) : `http://www.omg.org/mof/`, accessed 08/12.

In order to make this information usable for an entity ranking, we first abstract meta-data models and data models into *type graphs* and *entity graphs*; with entities referring to data objects and types to meta-data objects respectively. For this, we use directed acyclic graphs (DAGs) to describe entities, types and edges between entities and types representing "belongTo" relationships. We hereby explicitly distinguish between entities and types to facilitate the definition of separate ranking measures (see Section 3). Finally, we define a service advertisement as a set of entities linked to an ES. In the following, we formally describe (a) an entity graph, (b) a type graph, (c) a mapping to link entities to types and (d) a mapping to link signatures to entities.

Definition 1 (Type Graph G_C). *We define a directed acyclic type graph $G_C := (C, R_C)$ with C representing a set of types $c \in C$ and $R_C \subseteq C \times C$ denoting a set of directed edges between types.*

Definition 2 (Entity Graph G_E). *We define a directed acyclic entity graph $G_E := (E, R_E)$ with E representing a set of entities $e \in E$ and $R_E \subseteq E \times E$ denoting a set of directed edges between entities.*

Definition 3 (Entity-Type Mapping Φ). *We define a mapping $\Phi : E \to C$, $\Phi(e) = c$ for $e \in E$, with $\forall e \in E : \exists c \in C : \Phi(e) = c$. Furthermore, for each $c \in C$ we denote the (possibly empty) subset $E_c \subseteq E$ such that $\forall c \in C : \forall e \in E_c : \Phi(e) = c$. Obviously these subsets are distinct for different c, i.e. $\forall c_i, c_j \in C : c_i \neq c_j \Rightarrow E_{c_i} \cap E_{c_j} = \emptyset$*

Definition 4 (Signature-Entity Mapping Ψ). *We define a set of signatures $s \in S$ and a mapping $\Psi : S \to 2^E$, $\Psi(s) = E_D$ for $E_D \subseteq E$.*

Third, service design includes naming conventions for the purpose of consistency: they prescribe in which order(s) the types and entities mentioned above should be assembled when forming ES signatures. We use non-deterministic automata with ε-moves (NFA-ε) to represent naming conventions, describing a language of valid ESs signatures. For instance, the partial naming convention '`BO-BON`' expects type `BO` to be followed by type `BON` – e.g. '`SalesOrderItem`' (rather than '`ItemSalesOrder`'). Note that the construction of the NFA-ε can be done automatically by learning it from a sufficiently large set of existing ESs [17]. A formal definition and examples are given in [16].

3 Keyword-based Search using Entity Ranking

In this section, we describe the keyword-based search and entity ranking. Fig. 1 shows a flowchart of involved functionalities, i.e. (i) *Entity Detection*, (ii) *Entity Ranking* and (iii) *Entity Suggestion and ES Query* as described in the following.

3.1 Entity Detection

The entity detection function analyzes the free text query and identifies a list of completely and partially matched entities from the entity graph G_E. For this,

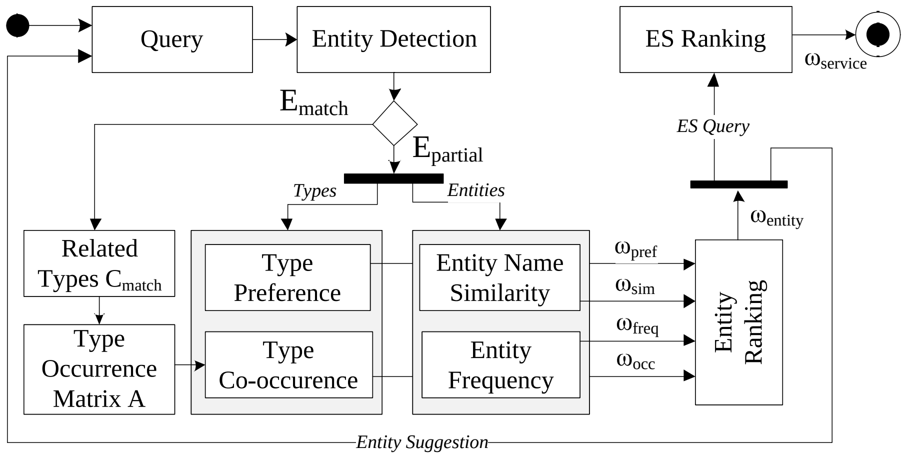

Fig. 1. Entity Suggestion and ES Query using Proposed Entity Ranking

the user input is first pruned to a list of noun and verbs, generally referred to as tokens. Each token is then normalized using the porter stemming algorithm[5]. We hereby understand the user input as a sequence of normalized tokens referred to as term string t. The term string t is then used to determine all possible token n-grams, i.e., token subsequences, using the following notation: $t = abc$, where a, b, and c are tokens. From t, we construct $\tilde{t} := \{\tilde{t}_1, \dots, \tilde{t}_6\} = \{a, b, c, ab, bc, abc\}$, where the $\tilde{t}_i$ are the n-grams from t. For instance, for $t \equiv$ `Sales Order` we have $\tilde{t} = \{$`Sales, Order, Sales Order`$\}$. In a final step, we check all $\tilde{t}_i \in \tilde{t}$ against the entity graph G_E for complete entity matches $E_{match} \subseteq E$ defined as follows:

$$E_{match}(t) := \{e \in E \mid \exists \tilde{t}_i \in \tilde{t} : \tilde{t}_i = e\} \qquad (1)$$

In a second step, we check all n-grams $\tilde{t}_i$ against the entity graph G_E to find partially matching entities. A partial match is a "sufficiently good" match between term n-grams $\tilde{t}_i$ and entity n-grams $\tilde{e}_j$, where $\tilde{e}_j$ is constructed from $e \in E$ as $\tilde{t}_i$ from t. This is expressed in a *entity similarity measure*, $\omega_{sim}(t, e)$, between term string t and entity e, which we formally define in Section 3.2. Roughly speaking, this similarity score is a normalized accumulation of the pairwise similarity of n-grams ($\tilde{t}_i$ and $\tilde{e}_j$), which is, in turn, calculated as the edit distance (or Levenshtein distance) [12] between the respective n-grams. The similarity is sufficiently high if $\omega_{sim}(t, e)$ exceeds a custom threshold ρ_{tr}. The set of partially matching entities $E_{partial}(t)$ for t thus is defined as

$$E_{partial}(t) := \{e \in E \mid \omega_{sim}(t, e) > \rho_{tr}\}. \qquad (2)$$

3.2 Entity Ranking

We propose an entity ranking as a combination of four different ranking measures: (i) an entity similarity ranking, (ii) an entity frequency ranking, (iii) a

[5] Porter Stemmer: `http://tartarus.org/martin/PorterStemmer`, accessed 08/12.

type preference ranking and (iv) a type co-occurrence ranking. The following subsections describe these ranking measures and how they are aggregated into a single ranking score for entities.

Entity Similarity In order to rank matches of terms and entities, we adopted the similarity ranking for terms used in [3]. This measure first computes a weight ω_q to describe how much of the term string t is covered by an n-gram $\tilde{t_i} \in \tilde{t}$:

$$\omega_q(\tilde{t_i}) := \frac{|\tilde{t_i}|}{|t|}$$

Second, the ranking calculates a similarity score ω_s between a term n-gram $\tilde{t_i} \in \tilde{t}$ and an entity n-gram $\tilde{e_j} \in \tilde{e}$ using the edit distance function $sim(\tilde{t_i}, \tilde{e_j})$:

$$\omega_s(\tilde{t_i}, \tilde{e_j}) := \frac{1}{\mathrm{sim}(\tilde{t_i}, \tilde{e_j}) + 1}(1 - \frac{\min(\mathrm{sim}(\tilde{t_i}, \tilde{e_j}), |\tilde{e_j}|)}{|\tilde{e_j}|})$$

The second factor represents the overall similarity of $\tilde{t_i}$ and $\tilde{e_j}$, which returns zero if the edit distance $sim(\tilde{t_i}, \tilde{e_j})$ exceeds the size of $\tilde{e_j}$. The first factor is used to reduce the weight of high edit distances, thus favouring shorter matches. The similarity value, $\omega_t(t, e)$, over all n-grams $\tilde{t}$ and $\tilde{e}$ is defined as follows:

$$\omega_t(t, e) := \sum_{\tilde{t_i} \in \tilde{t}} \omega_q(\tilde{t_i}) * \sum_{\tilde{e_j} \in \tilde{e}} \omega_s(\tilde{t_i}, \tilde{e_j})$$

Finally, we normalize the similarity score $\omega_{sim}(t, e)$ over similarity values ω_t:

$$\omega_{sim}(t, e) := \frac{\omega_t(t, e)}{\max(\{\omega_t(t, e_k) | e_k \in E\})} \tag{3}$$

Entity Frequency Second, we compute an entity frequency ω_{freq}. This is essentially "DF" from the standard TF/IDF ranking used in IR [12]: ω_{freq} disregards TF and basically reverses IDF, to capture how frequently signatures $s \in S$ link to a particular entity $e \in E$. As ES signatures have defined syntax, i.e., the naming conventions, they rarely contain entities of the same kind – making the TF part in TF/IDF obsolete. Also due to the clearly defined vocabulary, redundancy in terms is virtually non-existent – removing the need to prune them with IDF from TF/IDF. In contrast, we assign a higher importance to entities which are often linked to ES signatures. We argue that, for an entity-centric search, it is favorable to rank entities higher if more operations refer to them. We hereby contextualize the frequency of entities according to their type association, i.e. the frequency of an entity e in a corpus of ESs is normalized by the number of all ESs linked to an entity referring to the same type as e. To mitigate the effect of outliers, we apply a logarithm function (as defined in Sec. 2, Φ is the projection of entities to types, and Ψ the projection of signatures to sets of entities). Finally, we normalize the frequency score $\omega_{freq}(e)$ over all ω_f (analog to Eq. 3).

$$\omega_f(e) := \frac{\log(|\{s \in S \mid e \in \Psi(s)\}| + 1)}{\log(|\{s \in S \mid \forall e_i \in E, \Phi(e) = \Phi(e_i) : e_i \in \Psi(s)\}| + 1)}$$

Type Preference In contrast to previously described, entity-related ranking measures, the type preference uses a probabilistic distribution of preferred types of entities from existing ES search queries. For this, we reuse search logs collected from SAP over a period of three months, from which we identified user queries that contained exact matches of entities as defined in Eq. 1. In the following, we refer to the set of matched query term strings as T_q. Each query term $t_q \in T_q$ hereby contains at least one entity. Using T_q, we extract the matched entities, infer their associated types and aggregate the frequency of identical types. At this stage, we only consider exact matches of entities to reduce the ambiguity of associated types. We then define the type preference $\omega_{pref}(c)$ for a type $c \in C$ as the frequency of entities related to c, divided by total number of entities extracted from T_q. Finally, we use a logarithm function to mitigate outliers. Finally, we normalize the type preference score $\omega_{pref}(c)$ over ω_p (analog to Eq. 3).

$$\omega_p(c) := \frac{\log(|\{e \mid t_q \in T_q, e \in E_{match}(t_q), \Phi(e) = c\}| + 1)}{\log(|\{e \mid t_q \in T_q, e \in E_{match}(t_q)\}| + 1)}$$

Type Co-Occurrence The type co-occurrence $\omega_{occ}(t, c)$ determines the likelihood of a type c to occur with a set of types matched by the query term string t. Types are referred to by transitions in the corresponding automaton [17]. In that context, a type is considered co-occurring if it appears along an accepting path containing one or more types already matched to the user search query. To rank co-occurring types, we measure the frequency of their occurrence among all accepting paths, weighted by the number of contained matched types. The result of the ranking is a list of types which often appear with types identified in the user input. A detailed description of the type co-occurrence ranking component $\omega_{occ}(t, c)$ can be found in [15].

Combined Ranking Score To calculate a single ranking score for entities, we use a weighted average over all ranking measures. For this, we refer to the set of ranking measures as $\omega(t, e) \in \Omega(t, e) = \{\omega_{sim}(t, e),\ \omega_{freq}(e),\ \omega_{pref}(c = \Phi(e)),\ \omega_{occ}(t, c = \Phi(e))\}$. We define a weight function $p : \Omega \to \mathbb{R}$ for the ranking measures $\omega(t, e) \in \Omega(t, e)$: increasing the weight means increasing the relative importance of the respective ranking measure. The weighted average score $\omega_{entity}(t, e)$ is then computed for a query term string t and an entity $e \in E$:

$$\omega_{entity}(t, e) := \frac{\sum\limits_{\omega(t,e) \in \Omega(t,e)} p(\omega(t, e)) * \omega(t, e)}{\sum\limits_{\omega(t,e) \in \Omega(t,e)} p(\omega(t, e))} \tag{4}$$

3.3 Entity Suggestions & ES Queries

For the purpose of generating suggestions and querying related ESs, we refer to the workflow as shown in Figure 1. After the entity detection, we receive a (possibly empty) set of complete and partial entity matches E_{match} and $E_{partial}$. With E_{match}, we infer a set of related types and rank any co-occurring types using ω_{occ} and ω_{pref}. With $E_{partial}$, we use entity name similarity ω_{sim} and calculate their entity frequency ω_{freq}. Finally, we calculate the entity ranking score ω_{entity} (cf. Eq. 4) and use the list of ranked entities in two ways: to provide suggestions to the user, i.e., displaying the top-x-ranked entities; and to find and rank ESs. The latter requires an additional ranking score, to derive an ES ranking from the entity ranking. As such, we use the ratio of completely and partially matched entities to all entities associated to an ES. The score for each ES is the accumulation of the ranking scores of entities associated to the ES, divided by the total number of associated entities. Complete matches are counted as 1, partial matches as ω_{entity}, and unmatched entities as 0 (denoted as $val(t, e)$). We define the ranking score for an ES $s \in S$ and a query term t as follows:

$$val(t,e) = \begin{cases} 1, & \text{if } e \in E_{match} \\ \omega_{entity}(t,e), & \text{if } e \in E_{partial} \\ 0, & \text{otherwise} \end{cases}$$

$$\omega_{service}(t,s) := \frac{1}{|\Psi(s)|} * \sum_{e \in \Psi(s)} val(t,e) \tag{5}$$

4 Related Work

The search approach proposed in this paper can be seen as a combination of existing IR methods and additional domain knowledge – see e.g., [10] for an overview. For brevity, we omit some details here, which can be found in the TR. First, keyword-based searches can utilize additional knowledge in form of ontologies and/or linguistic knowledge to refine/expand free text queries and enhance the ranking of different types of documents, e.g., structured documents (e.g., Semantic Web [6, 20]) or unstructured documents (e.g., Web [2, 4, 3]). In contrast to our goals, these approaches require larger documents. However, some ranking measures in our work are inspired by approaches in this category, e.g., entity similarity ranking [3], entity ranking based on lexical or domain knowledge [2, 4], entity relationship-based ranking [1]. In a similar context, we consider the relationship defined in automata to rank types of entities and use entity frequency to identify key ESs. Second, our iterative querying paradigm is inspired by [13]. ActiveObjects [11] advocates the learning of actions on entities extracted from Web search logs – intriguing for Web queries, less applicable for service design with a well defined action vocabulary. QueryFeature-Graph [8] links queries to features of a system, e.g., captured from query logs, which could be used to further bridge the gap between user and system terminology. In the area of code search, Portfolio [14] provides a search for API functions using a combination of word similarity and word occurrences. Exemplar [9] describes an approach to enhance code search using API documentation. Neither makes use of models from model-driven engineering.

5 Conclusion and Future Work

We presented an iterative keyword search for ESs based on entities. As such, we proposed an entity ranking combining different ranking measures applied to entities and their associated types. The ranking returns a list of ranked entities, which we use as suggestions to the user as well as to find a set of relevant ESs related to these entities. Based on our user study, we conclude that such an entity-centric keyword search indeed increases the effectiveness of ES search: while the average number of search attempts increases slightly, precision and recall amongst the top ten search results increased steeply over a traditional IR method – for more details on the evaluation, we refer to [15]. In future work, we aim at (i) utilizing ES documentation in the search as another ranking measure; and (ii) deriving additional knowledge from previous searches.

References

1. B. Aleman-Meza, I. Arpinar, M. Nural, and A. Sheth. Ranking Documents Semantically Using Ontological Relationships. In *ICSC'10*.
2. A. B.-Jones, V. Storey, V. Sugumaran, and S. Purao. A Heuristic-Based Methodology for Semantic Augmentation of User Queries on the Web. In *ER*. 2003.
3. F. Brauer, M. Huber, G. Hackenbroich, U. Leser, F. Naumann, and W. M. Barczynski. Graph-Based Concept Identification and Disambiguation for Enterprise Search. In *WWW*, Raleigh, NC, USA, 2010. ACM.
4. J. Conesa, V. C. Storey, and V. Sugumaran. Improving Web-Query Processing Through Semantic Knowledge. *DKE*, 66(1):18–34, 2008.
5. F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The Next Step in Web Services. *Commun. ACM*, 46:29–34, October 2003.
6. L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: A Search and Metadata Engine for the Semantic Web. In *Conference on Information and Knowledge Management, CIKM'04*.
7. X. Dong and A. Halevy. Indexing dataspaces. In *ACM SIGMOD*, 2007.
8. A. Fourney, R. Mann, and M. A. Terry. Query-feature graphs: bridging user vocabulary and system functionality. In *UIST*, pages 207–216, 2011.
9. M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby. A Search Engine for Finding Highly Relevant Applications. In *ICSE '10*.
10. H. H. Hoang and A. M. Tjoa. The State of the Art of Ontology-based Query Systems: A Comparison of Existing Approaches. In *ICOCI'06*.
11. T. Lin, P. Pantel, M. Gamon, A. Kannan, and A. Fuxman. Active Objects: Actions for Entity-centric Search. In *WWW '12*.
12. C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge Univ. Press, 2008.
13. Y. Mass, M. Ramanath, Y. Sagiv, and G. Weikum. IQ: The Case for Iterative Querying for Knowledge. In *CIDR*, pages 38–44, 2011.
14. C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu. Portfolio: finding relevant functions and their usage. In *ICSE '11*.
15. M. Roy. *Facilitating Enterprise Service Management Using Service Design Knowledge*. PhD thesis, CSE, UNSW, 2013. (Under Review).
16. M. Roy, B. Suleiman, D. Schmidt, I. Weber, and B. Benatallah. Using SOA Governance to Augment Enterprise Service Descriptions. In *CAiSE'11*.
17. M. Roy, I. Weber, and B. Benatallah. Extending Enterprise Service Design Knowledge using Clustering. In *ICSOC'12*.
18. SAP. Enterprise Services Workplace, Aug. 2012. `http://esworkplace.sap.com`.
19. A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic. Searching the Web: The public and their queries. *JASIST*, 52(3):226–234, 2001.
20. E. Toch, A. Gal, I. Reinhartz-Berger, and D. Dori. A Semantic Approach to Approximate Service Retrieval. *ACM Trans. Inter. Tech.*, 2007.