Extending Enterprise Service Design Knowledge using Clustering

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

[1] SAP Research, Sydney, Australia
[2] School of Computer Science & Engineering, University of New South Wales
[3] Software Systems Research Group, NICTA, Sydney, Australia*
{m.roy, ingo.weber, boualem}@cse.unsw.edu.au

Abstract. Automatically constructing or completing knowledge bases of SOA design knowledge puts traditional clustering approaches beyond their limits. We propose an approach to amend incomplete knowledge bases of Enterprise Service (ES) design knowledge, based on a set of ES signatures. The approach employs clustering, complemented with various filtering and ranking techniques to identify potentially new entities. We implemented and evaluated the approach, and show that it significantly improves the detection of entities compared to a state-of-the-art clustering technique. Ultimately, extending an existing knowledge base with entities is expected to further improve ES search result quality.

1 Introduction

In large-scale software development efforts, such as enterprise Service-Oriented Architectures (SOAs), ESs are commonly developed using service design guidelines – guarded by SOA Governance [13, 8, 16]. These guidelines may include entities (e.g. Sales Order) and naming conventions used to construct unambiguous ES operation names, referred to as ES signatures. For instance, consider the ES signature "SalesOrderItemChangeRequestConfirmation_In" from SAP's ESR[4]. Although such service design knowledge is largely used to consistently design ES signatures, it can also be utilized for other applications, e.g. tools to automatically generate, duplicate-check and validate compliant ES signatures as well as to search for ESs. We tested the latter in an ongoing, separate stream of work, where we use service design knowledge in an entity-centric keyword search for ESs, with highly encouraging results. However, such service design knowledge can be incomplete, e.g. due to partial modeling, or become outdated as related systems and business requirements evolve over time. Particularly when customers tailor an off-the-shelf enterprise application to their specific need, they may disregard design guidelines when developing new ESs. Therefore, resulting ESs may not fully comply with existing service design guidelines and possibly incorporate new knowledge, i.e. new entities or additions to naming conventions.

* 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.

[4] Enterprise Service Registry: http://sr.esworkplace.sap.com

This knowledge is inherent to ES signatures and may not be reflected in service design knowledge yet. The absence of such knowledge can therefore limit the effectiveness of above mentioned applications, e.g. it strongly affects the precision and recall of search results in an entity-centric search. Hence, there is a need to extract design knowledge from existing ES signatures.

To amend existing knowledge bases, there are a few approaches that can be used such as entity recognition [3, 5, 9] or clustering techniques [7, 23, 12, 21]. With entity recognition, known entities can be recognized from text, e.g. using entity graphs [3]. The recognition effectiveness hereby also depends on the completeness of the entity graph and becomes uncertain when entities are missing. On the other hand, clustering techniques, e.g. hierarchical agglomerative clustering (HAC) [12, 7], can be used to find term clusters from text representing potentially new entities. Although these clusters can be used, e.g. for Web service matchmaking [7], it is not clear how well these clusters represent exact entity names. This is because clustering tries to assign terms to discrete clusters, which does not work well for entities consisting of repetitive terms overlapping with other entities, e.g. "Sales Order" and "Sales Price". Also, clustering assumes potential entities to be statistically independent. In that context, we observed that naming conventions may cause terms of entities to frequently co-occur with each other, which clustering can misunderstand as a cluster.

In this paper, we propose an approach that combines entity recognition and a knowledge base-driven clustering to find names of unknown entities from ES signatures. We hereby aim to improve the accuracy of recognizing new entities by removing known entities from the input to our knowledge base-driven clustering, and by possibly merging newly formed clusters with co-exiting entities. First, we utilize existing service design knowledge to learn existing naming conventions. Second, we reuse naming conventions to recognize known entities in ES signatures. Third, we perform the knowledge base-driven clustering over the remaining, unknown terms and check their co-occurrence with recognized entities. For this, we introduce measures of confidence and cohesion, to describe the quality of (potentially overlapping) term clusters and the strength of their connectivity to co-occurring entities. Finally, resulting clusters are added to the knowledge base, as either new or specialized entities.

We implemented the proposed approach and evaluated the knowledge base-driven clustering compared to the HAC used in [7] on a large-scale repository from SAP with more than 1600 ES signatures. Our evaluation shows that the proposed approach achieves reasonably high precision and recall values of clustered entities and outperforms the HAC. In short, our contributions are as follows:

- Reusing service design knowledge to recognize entities from signatures
- A knowledge base-driven clustering approach, which uses generated (potentially overlapping) term clusters and recognized entities to find new entities.
- An in-depth evaluation using a real-world testbed.

We next describe the knowledge base and challenges in its effective use. In Section 3, we explain the proposed approach, followed by the evaluation in Section 4. Related work is discussed in Section 5, Section 6 concludes the paper.

2 Using Service Design Knowledge

We start by motivating and describing an abstract representation of service design knowledge, followed by open challenges how to effectively use the knowledge.

2.1 A Representation of Service Design Knowledge

Organizations use SOA Governance to better manage their SOA [13], which can be applied to any part of a service life-cycle addressing areas such as service design and development among others. In this work, we only focus on the service design phase [2]. In this phase, enterprises employ service design methodologies to create business-aligned, reusable and long-living ESs [16, 8]. Such methodologies typically describe guidelines and best practises providing clear instructions to developers on how to create and name services that comply with agreed-on design principles. These design principles are the basis of our knowledge base, as described in detail in previous work [15]. There we also showed how to derive an abstract representation of this service design knowledge consisting of (i) a graph of entities and (ii) an automaton describing a set of ES signatures. This representation is summarized below.

First, we define a Directed Acyclic Graph (DAG) of typed entities $e \in E_{KB}$ with type $c \in C_{KB}$ in RDF. This DAG captures an abstract view of a data model and related design patterns, stemming from a service design methodology. Fig. 1 shows a partial example DAG, depicting entities e_i as white boxes (i.e. RDF literals) and associated types c_j as grey ovals (i.e. RDF classes). For instance, entity e_{14} : `Sales Order` is of type c_9 : `Business Object`; e_{14} belongs to entity e_{15} : `Sales Order Processing` of type c_{10} : `Process Component`. We consider such a DAG as a structured vocabulary of typed entities.

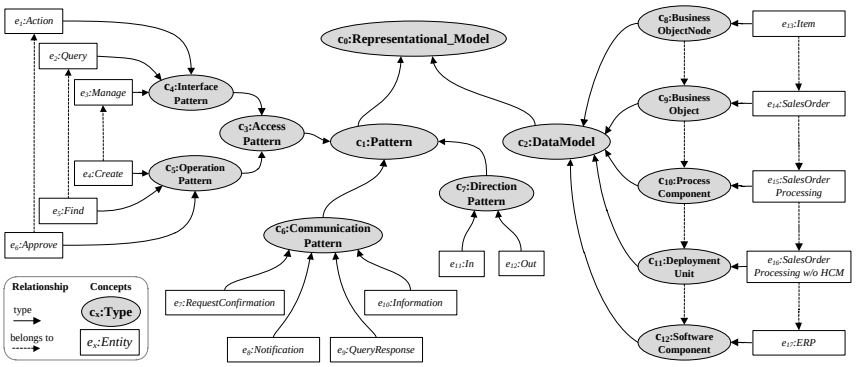


Fig. 1. Example of a typed entity graph representing a structured vocabulary

Second, we define a non-deterministic automata with epsilon moves to capture a (possibly incomplete) set of naming conventions. We defined the automaton on a set of entities, the input alphabet. The set of transitions uses the types of entities. As such, an ES signature $S_i \in S$ is interpreted as a sequence of entities e_i, where each respective type c_j triggers a state transition. A governance-compliant signature is accepted by the corresponding automaton, if it reaches a

final state in the automaton after its last entity. Fig. 2 shows an excerpt of an automaton, related to the example DAG in Fig. 1.

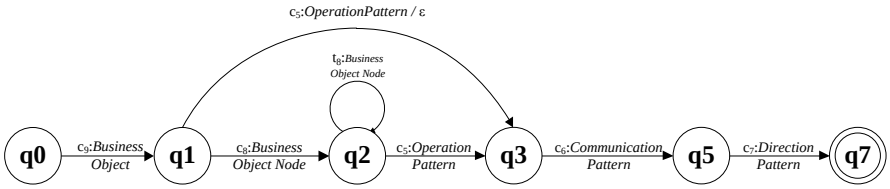


Fig. 2. Example of an automaton describing a subset of ES signatures

The following example ES signature, S_{x_1}, is accepted by the automaton from Figure 2, according to the split into entities of (abbreviated) types shown below.

$$S_{x_1}: \underbrace{\texttt{SalesOrder}}_{c_9:\texttt{BO}}\underbrace{\texttt{Item}}_{c_8:\texttt{BON}}\underbrace{\texttt{Change}}_{c_5:\texttt{OP}}\underbrace{\texttt{RequestConfirmation}}_{c_6:\texttt{CP}}\underbrace{\texttt{\_In}}_{c_7:\texttt{DP}}$$

2.2 Challenges of Using Service Design Knowledge

In the previous section we provided an example of an ES signature that completely matched exact entities in the knowledge base and is accepted by a respective automaton. Although these types of ES signatures constitute the majority of cases, there are signatures that can only be matched partially as shown in the example below[5]. Apart from recognized entities, the signature also contains terms (here: `Reporting` and `Bulk`) that cannot be matched to entities. In this context, we refer to terms as single words separated by their camel case notation.

$$S_{x_2}: \underbrace{\texttt{Reporting}}_{?}\underbrace{\texttt{Employee}}_{c_9:\texttt{BO}}\underbrace{\texttt{Bulk}}_{?}\underbrace{\texttt{Notification}}_{c_8:\texttt{BON}}\underbrace{\texttt{ByID}}_{c_5:\texttt{OP}}\underbrace{\texttt{QueryResponse}}_{c_6:\texttt{CP}}\underbrace{\texttt{\_In}}_{c_7:\texttt{DP}}$$

Since these terms do not match existing entities, they might be (parts of) entities missing from the KB. Using signature S_{x_2} as an example, we introduce two types of new entities as follows. First, the term `Reporting` alone does not seem to represent an independent entity. Instead, `Reporting` together with the entity following it, i.e. `Employee`, describes a specialization of the existing `Employee`. In contrast, the term `Bulk` is not a specialization of `Notification`, but rather a general property: `Bulk` is also found in the context of other entities, e.g. `Payment` or `Message`. In the following section, we describe an approach that recognizes known entities, extracts unknown terms and determines whether they are likely a specialization of an existing entity or a separate, new entity.

3 Extending Services Design Knowledge using Clustering

In this section, we describe the proposed solution to identify unknown entities from ES signatures. We start with an overview, before explaining each step of the approach in detail. The inputs to the approach are a list of ES signatures and a populated entity graph (cf. Section 2). Given this input, the following four phases are executed (see Fig. 3).

[5] we used a constructed example of a signature to illustrate two common pitfalls

Strict Classification recognizes signatures that are completely matched with entities from the knowledge base. For each of these signatures, it extracts the corresponding entity sequence and consolidates all sequences into an automaton using [19].

Approximate Classification uses the automaton from the first step to detect known entities in the remaining signatures. All unknown terms are collected.

Knowledge base-driven Clustering applies two measures to candidate entities: (i) a confidence measure to form term clusters of closely connected unknown terms and (ii) a cohesion measure to merge term clusters with already recognized entities. If predefined thresholds are exceeded, a cluster can be considered as a new entity. In contrast to common clustering methods which only take into account unknown terms, our knowledge base-driven clustering uses the existing KB in addition to the unknown terms.

KB Extension adds new entities to the KB: pure term clusters represent separate new entities; term clusters merged with existing entities represent specialization of these entities.

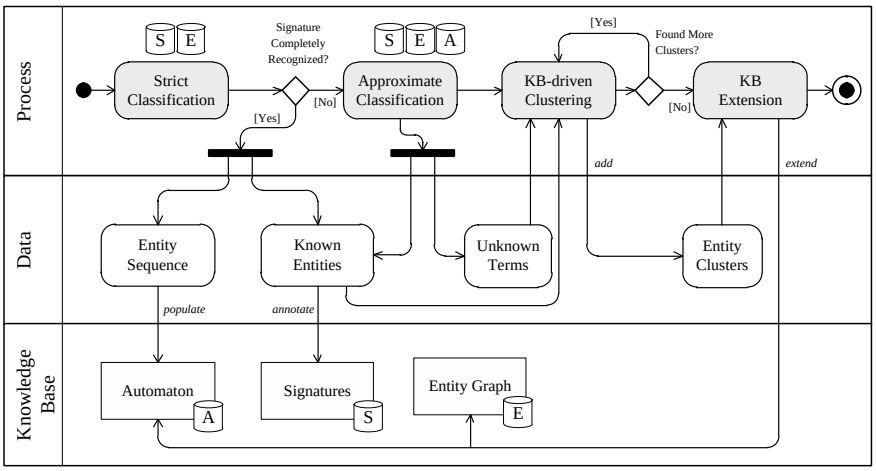


Fig. 3. Overview of the different components of the proposed approach

The strict and approximate classification can be considered as a pre-clustering of signatures $S_i \in S$ into a set of recognized entities $E_i \in E$ and a set of unknown terms $T_i \in T$. In the remainder, we therefore refer to a signature S_i as a 3-tuple of E_i, T_i, and O_i:

$$S_i := (E_i, T_i, O_i) \tag{1}$$

where O_i specifies the order of terms and entities: $O_i = (x_{i_1}, ..., x_{i_m})$ where $x_{i_j} \in E_i \cup T_i$. A pair of signature parts $x_{i_j}, x_{i_k} \in E_i \cup T_i$ is said to be *neighbors in* S_i iff $j = k+1$ or $j = k-1$. A set of signature parts $X = \{x_j, ..., x_k\}$ with $X \subseteq E_i \cup T_i$ is said to be *connected in* S_i iff there is a permutation $X' = (x_{j_1}, ..., x_{j_n})$ of X such that all pairs $(x_{j_l}, x_{j_{l+1}})$ are neighbors in S_i.

3.1 Strict Classification

The goal of this phase is to reverse-engineer approximate naming rules from signatures that can be explained fully and unambiguously. For this, the strict classification uses as input the set of ES signatures (S) and an entity graph (E) as depicted in Figure 3. First, it identifies matches between all parts of a signature and the vocabulary defined in the entity graph. If not every part of a signature matches some entity from the vocabulary, the signature is rejected.

For the remaining signatures, there may be more than one explanation – i.e. more than one set of known entities that together matches all parts of the signature. To reject signatures with unclear explanations, the classification uses the relationships from the entity graph as follows. For the recognized entities, it traverses the ancestors and descendants from the "belongs-to" relationship, and builds sets of *graph-related* entities and types respectively. It then requires that all entities in the signature whose types are graph-related, are themselves graph-related as well. For instance, the signature "SalesOrderItemCreateRequestConfirmation_In" contains the entity "Item" (typed "Business Object Node" (BON)) and the entity "Sales Order" (typed "Business Object"(BO)). Due to BON being graph-related to BO (see Fig. 1), "Item" is also required to be graph-related to "Sales Order" – which is indeed the case. If, for any pair of entities in an explanation, this constraint does not hold, the classification rejects the explanation. Note that this is, in practical terms, very strict indeed: some correct explanations are likely to be rejected. Finally, the strict classification rejects any signature where the number of remaining explanations is not exactly 1.

The result of the strict classification is a set of signatures, each explained by sequence of recognized entities. These entity sequences are then used to build a minimal automaton as described in [19], and annotate related signatures with the recognized entities [15]. Inversely, the automaton transitions are annotated with their respective *popularity*, i.e., the absolute number, of how often a given transition has been used in the strictly classified signatures.

Example 1. Strict Classification result for S_{x_1}:

$$S_{x_1} = (E_{x_1}, \emptyset)$$
$$E_{x_1} = \{\texttt{SalesOrder, Item, Change, RequestConfirmation, In}\}$$
$$O_{x_1} = (\texttt{SalesOrder, Item, Change, RequestConfirmation, In})$$

3.2 Approximate Classification

The approximate classification takes as an input the ESs signatures (S), the entity graph (E), and the automaton (A) built in the previous step. For each signature that has been rejected by the strict classification, the approximate classification aims to find known entities that explain parts of it.

As in the previous step, approximate classification has to deal with the challenge of multiple possible explanations for signature parts. For example, the compound term "Sales Order Confirmation" could be considered (a) a single entity (typed BO) or (b) as a concatenation of two independent entities "Sales Order" (typed BO) and "Confirmation" (typed "Communication Pattern" (CP)). The

approximate classification therefore uses the automaton to find the most likely explanation amongst multiple possibilities.

This phase starts off like the previous one, by matching neighboring signature parts against entities from the entity graph, to produce a set of possible explanations (if any) for the parts of each signature. For signatures with multiple explanations, the approximate classification filters and ranks the explanations according to their level of match and popularity with the automaton. That is, explanations that are contradictory to the automaton are rejected. For the remaining explanations, the classification computes the sum of the popularity of the transitions taken by this explanation, and ranks the remaining explanations accordingly. It then rejects all but the highest-ranked explanation.

As an example for the filtering, the automaton may denote that a valid signature only contains one CP. Therefore, using the above partial signature example "Sales Order Confirmation", explanation (b) would be rejected if another CP appears later in the signature and explanation (a) is kept. As an example for the ranking, say there was no other CP. Then, say explanation (a) might have a partial popularity of 320, and explanation (b) a partial popularity of 320 + 80 = 400, the ranking would prefer explanation (b). Finally, we collect recognized entities E_i and remaining unknown terms T_i to be used in the next step.

Example 2. Approximate Classification result for S_{x_2}:

$S_{x_2} = (E_{x_2}, T_{x_2}, O_{x_2})$

$E_{x_2} = \{$`Employee, Notification, By, ID, QueryResponse, In`$\}$

$T_{x_2} = \{$`Reporting, Bulk`$\}$

$O_{x_2} = ($`Reporting,Employee,Bulk,Notification,By,ID,QueryResponse,In`$)$

3.3 Knowledge base-driven Clustering

The goal of the knowledge base-driven Clustering is to find potential clusters of entities from the set of unknown terms T_i, in the context of existing sets of entities E_i and the order O_i. For this we use a confidence and a cohesion measure, as illustrated in Figure 4. In summary, the clustering first groups terms that co-occur in multiple signatures into term clusters (cf. right dashed bounding box), and computes the *term cluster confidence* score *conf* of the cluster candidates. Second, it merges term cluster candidates that have a high *conf* value with co-occurring entities using the *entity cohesion* score *coh*: the cohesion between term clusters and entities (cf. left and right dashed bounding boxes). We now explain both measures in detail.

Term Cluster Confidence. To identify potential term clusters, we consider any non-empty subset of terms $T_i \subseteq T$, as long as some signature contains this subset (formally: $\exists S_j : T_i \subseteq T_j$) and the terms in T_i are connected in S_j (as per the above definition). We refer to the set of all potential term clusters fulfilling this condition as $\mathcal{T}$. To calculate a confidence value $conf(T_i)$ for an arbitrary, but fixed term cluster T_i, we first define $S_t \subseteq S$ as the set of signatures whose respective term set T_j is a superset of T_i:

$$S_t(T_i) := \{S_j \in S \mid T_i \subseteq T_j, S_j = (E_j, T_j, O_j)\} \tag{2}$$

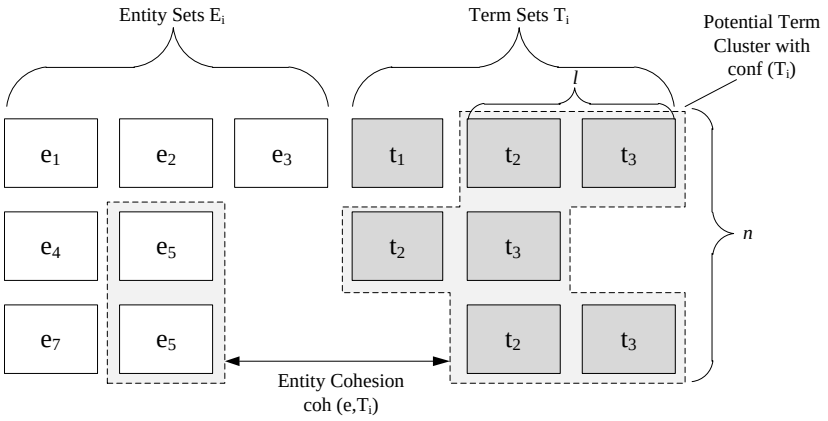


Fig. 4. Schematic clustering example, using Term Confidence and Entity Cohesion. Not shown is the influence of the term/entity order.

We further define p_n as the size of S_t (the number of signatures containing T_i), p_l as the size of T_i, and p_r as the average size of the complete term sets contained in S_t:

$$p_n(T_i) := |S_t| \tag{3}$$

$$p_l(T_i) := |T_i| \tag{4}$$

$$p_r(T_i) := \frac{\sum_{j:S_j \in S_t} |T_j|}{p_n(T_i)} \tag{5}$$

Finally, we define the term cluster confidence *conf* for T_i, normalized over all term cluster candidates $\mathcal{T}$ where $a, b \in [1, \infty)$ are decimal values:

$$p_t(T_i) := \ln(p_n(T_i)) \times \frac{p_l(T_i)^a}{p_r(T_i)^b} \tag{6}$$

$$conf(T_i) := \frac{p_t(T_i)}{\max(\{p_t(T_k) | T_k \in \mathcal{T}\})} \tag{7}$$

The fraction p_l/p_r hereby represents the ratio of the sizes of the term cluster vs. the average size of all term sets in the signatures in which T_i occurs. This fraction can be seen as a signal-to-noise ratio T_i in relation to all unknown terms. We further exponentiate the numerator of the fraction by a and the denominator with b. Intuitively, setting higher values for a prioritizes long term clusters. In contrast, setting higher values for b increases the penalty for long "rests", i.e., unknown terms that are not part of the candidate cluster. Since p_r is the average number of all unknown terms, i.e., the rest as well as the cluster, we recommend setting $a \geq b$ – otherwise longer clusters would in general be penalized. Based on samples, we found that $a = 3$ and $b = 2$ returns best results. The third main factor in p_t is p_n, the number of occurrences of the cluster candidate. To mitigate imbalance due to high variance in p_n, we apply the natural logarithm function. Preference between p_n and p_l/p_r can be expressed with the ratio a/b: the higher,

the less important p_n becomes. In summary, the confidence score prioritizes long clusters, appearing often, but with little noise.

Next, we define a threshold tr_{conf}: after calculating the confidence values for all candidate clusters, all term cluster candidates T_i with $conf(T_i) > tr_{conf}$ are added to the set of resulting clusters. This threshold primarily works by allowing *rivaling* explanations of term clusters to get added to the result set *simultaneously*. The whole confidence computation step is run as a *fixpoint algorithm*: term clusters that are added to the result set in one round are removed from the set of unknown terms. This results in a changed set of cluster candidates. For these we re-calculate *conf*, and so on. This is done until no more clusters are found. Since *conf* is normalized among the *current* set of candidates, at least the cluster with maximal confidence is added to the result set.

Finally, we *filter* the term cluster result set with a two-dimensional threshold $tr_f := (tr_{f1}, tr_{f2})$ as follows. tr_{f1} is a relative threshold: clusters in the result set are ranked according to their p_t value, and the lowest-ranked portion of size tr_{f2} are removed – e.g., the last 10%. In contrast, tr_{f2} is an absolute threshold, with respect to p_t: all term clusters T_i with $p_t(T_i) \leq tr_{f2}$ are removed from the result set. The combined threshold works to remove noise from the result set.

Entity Cohesion. As motivated with the example of "Reporting" and "Employee" above, some term clusters (e.g., "Reporting") should not form entities by themselves, but should be merged with other entities to form a specialization of those (e.g., "ReportingEmployee").

To determine which cluster candidates from the previous step should be merged with some entity, we compute a cohesion score *coh* that captures co-occurrences of connected term clusters T_i with entities. For this, we first define the projection $\rho(T_i)$ as the set of entity sets E'_j that (i) co-occur with T_i in some signature E_j, and (ii) contain only entities neighboring with some term from T_i in S_j. We further define the entity set U_c as the union of these entity sets.

$$\rho(T_i) := \{E'_j \mid T_i \subseteq T_j, S_j = (E_j, T_j, O_j), E'_i \subseteq E_j, \\ \forall e \in E'_j \; \exists t \in T_i : t, e \text{ are neighbors}\} \tag{8}$$

$$U_c := \bigcup_{E'_j \in \rho(T_i)} E'_j \tag{9}$$

We then define the cohesion $coh(e, T_i)$ between an entity e and a term cluster T_i as the ratio between signatures containing both e and T_i and all signatures containing T_i:

$$coh(e, T_i) := \frac{|\{E'_k \mid e \in E'_k, T_i \subseteq T_j, E'_k \in \rho(T_j)\}|}{p_n(T_i)} \tag{10}$$

Note that *coh* uses the projection defined above, and thus only counts co-occurrences between e and T_i where e is the neighbor of some term in T_i.

After calculating the cohesion between T_i and all candidate entities e, we decide if and how to combine T_i and the candidates as follows. If each occurrence of T_i is a co-occurrence with e (formally: $coh(e, T_i) = 1$), we merge T_i with e and add the result as a specialization of e. If for each candidate entity e we have

$coh(e, T_i) < 1$, a threshold tr_{coh} is applied to determine if the co-occurrence between e and T_i is frequent enough to justify merging. If the cohesion exceeds tr_{coh} merging is justified – however, there must be cases where the term cluster appears without e (else the cohesion would have been 1). Therefore, we add both the term cluster by itself, as a new entity, and term cluster merged with entity to the knowledge base. The merged cluster hereby becomes a specialization of e as well as the new entity. If the cohesion is below the threshold, we only add the original term cluster to the knowledge base as a new entity. The clustering result $C(e, T_i)$ between e and T_i can thus be formally defined as:

$$C(e,T_i) := \begin{cases} e \cup T_i & , \forall e \ \in U_c : coh(e,T_i) \ = 1 \\ \{T_i, e \cup T_i\} & , \nexists e' \in U_c : coh(e',T_i) = 1 \\ & \quad \wedge \forall e \in U_c : tr_{coh} < coh(e,T_i) < 1 \\ T_i & , \nexists e' \in U_c : tr_{coh} < coh(e,T_i) \leq 1 \\ & \quad \wedge \forall e \in U_c : coh(e,T_i) < tr_{coh} \end{cases} \tag{11}$$

3.4 Knowledge Base Extension

The outcome of the clustering are cluster sets – independent, new entities and new entities as specializations of existing entities. Depending on the type of cluster, different implications apply regarding the extension of the entity graph and automata with clustered entities as follows.

Adding Entities to the Entity Graph. Term clusters that have not been merged with existing entities only consists of terms and are therefore treated as independent entities. Every independent entity gets assigned a new, specific type, which is added directly under the root node in the entity graph. This is done because there is insufficient information to determine if the new entity should be considered to be of an existing entity type. In contrast, a specialization of an entity is added to the entity graph as a new entity under the same parent as the original entity and gets assigned the same type of the original entity.

Adding Entities to the Automaton. Since the automaton uses entity types as it alphabet, each newly added independent entity requires a new state to be added to the automaton. The corresponding new type is added as a new transition. Figure 5 shows this for our example. Since specialized entities inherit the type of the original entity, the transition already exists and no changes have to be made to the automaton. Note that the update of the automaton is done automatically during the next strict classification using the revised entity graph.

Example 3 (Knowledge Base Extension Example).
Say, term clusters $T_b = \{\texttt{Bulk}\}$ and $T_r = \{\texttt{Reporting}\}$ both exceed given thresholds tr_{conf} and tr_f, and that T_r plus entity `Employee` exceeds a given tr_{coh}. Therefore, T_b is added to the entity graph as a new entity e_b = `Bulk` with equally-named type c_{13} = `Bulk`. During the next iteration, e_b will be recognized as an entity and incorporated into the automaton (cf. Figure 5). In contrast, term cluster T_r will be merged with entity `Employee` and added to the knowledge base as entity e_r = `Reporting Employee` with the same type as `Employee`, i.e. c_9 : `Business Object`. During the next iteration, e_r will be recognized as an entity supporting the already existing transition based on type c_9.

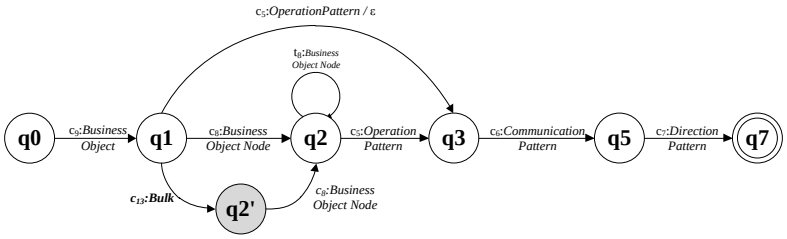


Fig. 5. Extending the example service design automaton from Fig. 2

As a result, the revised automaton also accepts signature S_{x_2} as follows:

$$S_{x_2}: \underbrace{\texttt{Reporting Employee}}_{c_9:\texttt{BO}}\ \underbrace{\texttt{Bulk}}_{c_{13}:\texttt{BULK}}\ \underbrace{\texttt{Notification}}_{c_8:\texttt{BON}}\ \underbrace{\texttt{ByID}}_{c_5:\texttt{OP}}\ \underbrace{\texttt{QueryResponse}}_{c_6:\texttt{CP}}\ \underbrace{\texttt{In}}_{c_7:\texttt{DP}}$$

4 Evaluation

We implemented the proposed, knowledge base-driven clustering approach $\mathcal{A}_{kb}$. As a baseline, we also implemented the HAC approach $\mathcal{A}_{cl}$ used in [7], and evaluated the performance of both approaches on a real-world testbed.

4.1 Evaluation Setup

The performance evaluation is based on a corpus of 1651 of SAP's (A2X) Enterprise Services, as well as an entity graph (cf. Figure 1) which we extracted from SAP's Enterprise Service Registry. Using the set of signatures and the entity graph as an input, we first performed a strict classification (cf. Section 3.1). The list of completely recognized signatures is from here on referred to as S_t, the set of contained entities as E_t. We use S_t and E_t as the basis to determine precision, i.e. the fraction of clustered entities that is correct, and recall, i.e. the fraction of correct entities that has been clustered, for $\mathcal{A}_{kb}$ and $\mathcal{A}_{cl}$. Finally, we conducted the evaluation process as shown in Figure 6.

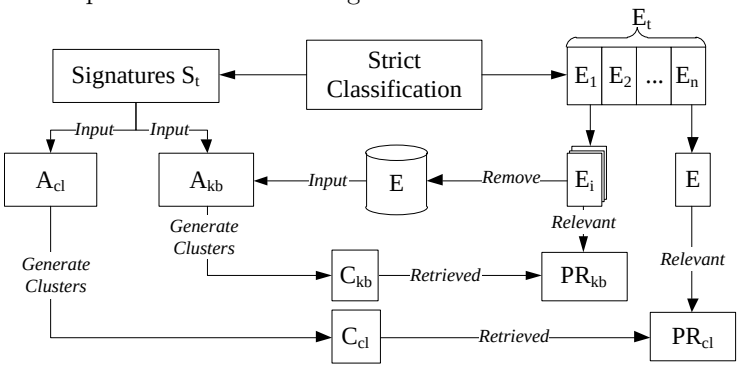


Fig. 6. Evaluation Setup for $\mathcal{A}_{kb}$ and $\mathcal{A}_{cl}$

Knowledge Base-driven Clustering $\mathcal{A}_{kb}$. For $\mathcal{A}_{kb}$, we used the set of signatures S_t and a *shortened version* of the entity graph as an input. The entity graph is reduced by removing a subset of entities from E_t, so that when we fed

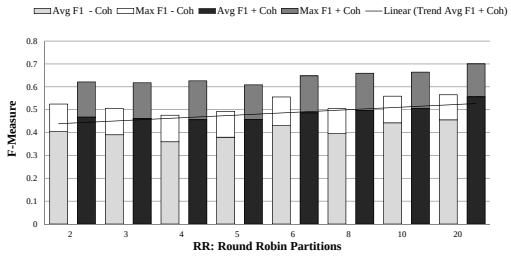
it into $\mathcal{A}_{kb}$, we can check if it produces clusters that match the entities removed before. By doing so, we can realistically simulate the situation where an incomplete KB has to be extended, while being able to check with certainty which clustering results are correct. For this, we apply a *round robin* strategy to determine equally-sized random subsets of entities $E_i \in E_t$, which are iteratively removed from the entity graph. The so-obtained shortened entity graphs are fed to $\mathcal{A}_{kb}$. For instance, a round robin partition of 3 initiates three different iterations, each having a third of E_t randomly removed from the entity graph and the result fed into $\mathcal{A}_{kb}$. By comparing the outputted clusters from $\mathcal{A}_{kb}$ against the previously removed E_i, we can calculate the average precision and recall PR_{kb} over all partitions. As usually, the F-Measure *F1* is the harmonic mean between PR. Using different numbers of round robin partitions allows us to investigate clustering performance in relation to how much of the entity graph is missing – i.e., how incomplete the KB is to begin with. In our experiment, we used the round robin partitions $RR \in [2, 3, 4, 5, 6, 8, 10, 20]$, which we each executed three times to mitigate randomness-based effects. For each round robin partition, we ran different combinations of the thresholds for confidence $tr_{conf} \in [0.1, \ldots, 0.9]$ and entity cohesion $tr_{coh} \in [0.1, \ldots, 0.9]$. Due to the way in which we created the test cases (all signatures in S_t were completely recognized before removing some entities), there is no noise present in the input signatures S_t. Hence, we switched the noise-filtering off by setting $tr_f = (0, 0)$.

Hierarchical Agglomerative Clustering $\mathcal{A}_{cl}$. For $\mathcal{A}_{cl}$, we used the HAC described in [7], which only requires a list of signatures S_t as an input. As in the strict classification, we split signatures in S_t into terms using their camel case notation. We then ran the term clustering with different values of support $p_s \in [1, \ldots, 10]$ (occurrences of a term – see [7] for details) and confidence $p_c \in [0.1, \ldots, 0.9]$ (co-occurrence of two terms). The outcome is a set of term clusters C_{cl}. As for $\mathcal{A}_{kb}$, we determined precision and recall PR_{cl}, but here based on C_{cl} and the complete set of entities E_t used in S_t.

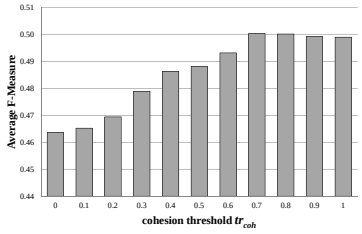
4.2 Evaluation Results

First, consider the influence of entity cohesion. Figure 7(b) depicts the average F-Measure for various cohesion thresholds tr_{coh}. As shown, the average F-Measure steadily increases until its maximum at $tr_{coh} = 0.7$ and only slightly decreases beyond that. Figure 7(a) shows average F-Measures (*AvgF1*) and maximum F-Measure (*MaxF1*) for $\mathcal{A}_{kb}$ without (*–Coh*) and with cohesion (*+Coh*). With cohesion, *AvgF1* and *MaxF1* are continuously higher than without: F-Measure increases with cohesion by $11\% - 21\%$ (*AvgF1*) and by $14\% - 24\%$ (*MaxF1*). The graph also shows that $AvgF1 + tr_{coh}$ steadily increases with growing round robin partitions, as illustrated by the trend line. Therefore, the highest results can be found at $RR = 20$, with an average *F1* of 0.55 and maximum *F1* of 0.70.

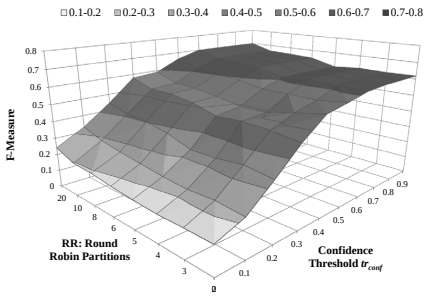
Figure 8(a) and 8(b) show the F-measure of $\mathcal{A}_{kb}$ and $\mathcal{A}_{cl}$ for combinations of $tr_{conf} \times RR$ (with a fixed $tr_{coh} = 0.7$) and $p_c \times p_s$ respectively. From Figure 8(a), it becomes clear that the term confidence tr_{conf} significantly influences the effectiveness of the $\mathcal{A}_{kb}$ clustering. That is, the F-Measure significantly increases with growing confidence thresholds and plateaus around its maximum



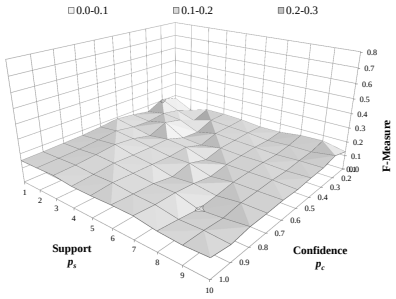
(a) Avg and Max F-Measure for RR



(b) Avg F-Measure for tr_{coh}



(a) F-Measure for $\mathcal{A}_{kb}$



(b) F-Measure for $\mathcal{A}_{cl}$

Fig. 8. Overall effectiveness of $\mathcal{A}_{kb}$ and $\mathcal{A}_{cl}$ for varying parameters

of ≈ 0.7 for $tr_{conf} \geq 0.6$. The graph also shows a slightly higher F-Measure with increasing round robin partitions, and reaches a local maximum (relative to tr_{conf}) at $RR = 20$. Hence, we found the globally maximal F-Measure of ≈ 0.7 for $RR = 20$, $tr_{conf} \geq 0.6$ and $tr_{coh} = 0.7$. Figure 8(b) shows the results for the $\mathcal{A}_{cl}$ clustering. The F-Measure here ranges from 0.1 to a maximum of 0.21 ($p_c = 0.2$ and $p_s = 1$). It also shows that the parameters p_c and p_s do not have a significant impact on the overall performance of the clustering. From close observation of the data, we found that, when increasing p_c and p_s, infrequent but relevant terms are ignored, thus affecting both precision and recall. On the other hand, smaller values for p_c and p_s increase the number of terms considered by $\mathcal{A}_{cl}$ to find more term clusters – relevant and irrelevant alike. Therefore, an increase of recall might largely be outweighed by a decrease of precision.

5 Related Work

Since the proposed approach combines entity recognition with clustering, we mainly focus on prior work related to these two areas.

Named Entity Recognition Using Entity Graphs. Although most Named Entity Recognition (NER) references can be found with unstructured text, described recognition techniques, e.g. based on similarity scores [3] might be relevant. For instance, [9, 20, 3] describe graph based identification of entities using entity (and relationship) similarity measures. In [5], the authors calculate a similarity score of overlapping segments of text with entities from dictionaries. Similar to [4], they use a TF-IDF similarity score [22] and assume statistically

independent terms, which is not the case for compact ES signatures where terms appear in a specific order. Moreover, for such structured signatures, a similarity score seems to be less feasible. That is, potential entities are found as exact matches during strict/approx. classification or as new entities during clustering.

Named Entity Recognition Using Supervised Learning. In general, rule-based approaches can be applied to extract named entities from documents, which often requires a significant manual effort. Therefore, supervised machine learning techniques [6, 17] and bootstrapping algorithms [1, 11] have been proposed to recognize entities or classify, e.g. Web Services [10, 14]. For instance, [11] describes a similar approach based on a three level bootstrapping algorithm used with machine learning for regular expressions. The difference to our work is that no upfront naming definitions are required as they are extracted from unambiguous signatures - for multiple entity types. Further, we do not extract entities similar to recognized entities but compliant to recognized naming definitions.

Unsupervised Clustering. Many clustering approaches have been proposed using e.g. document similarity to merge cluster at document level. For instance, single and complete link [18] use the smallest minimum and maximum pair-wise similarity distance between two clusters. Word-IC [23] uses a hierarchical agglomerative clustering (HAC) [21] based on the number of intersection terms within documents. Our work differs that we use clustering on operation-level to cluster entities (rather than documents) and thus can better determine the accuracy of intersecting terms relative to their noise. In [7, 12] the authors also use HAC, e.g. based on term co-occurrences in operation names [7], to measure the cohesion and correlation within and between term clusters. However, we showed that such clustering performs moderate for entities with overlapping terms whose appearance is not statistically independent (e.g. using naming conventions).

6 Conclusion

In this paper, we presented an approach for amending incomplete knowledge bases of ES design knowledge. The approach only requires a set of ES signatures and an incomplete KB. It first reverse-engineers naming conventions and uses them to filter out unlikely explanations for partially understood signatures. The approach then suggests term clusters from unknown parts of the signature and possibly merges them with co-existing entities to form new entities. We evaluated the approach on a testbed of 1651 ES signatures from SAP. After removing parts of the KB, we tested how successful the approach re-added missing parts. The approach performed reasonably well when half of the KB was removed, and even better when only smaller chunks were missing. Moreover, the approach performed significantly better than a state-of-the-art clustering approach. In future work, we plan to improve our search engine for ESs and extend the current comparative experiment to other clustering techniques.

References

1. E. Agichtein and L. Gravano. Snowball: Extracting Relations From Large Plain-Text Collections. DL '00, pages 85–94, New York, USA, 2000. ACM.

2. S. G. Bennett, C. Gee, R. Laird, A. T. Manes, R. Schneider, L. Shuster, A. Tost, and C. Venable. *SOA Governance: Governing Shared Services On-Premise and in the Cloud*. Prentice Hall, 2011.
3. F. Brauer, M. Huber, G. Hackenbroich, U. Leser, F. Naumann, and W. M. Barczynski. Graph-Based Concept Identification And Disambiguation For Enterprise Search. WWW '10, pages 171–180, New York, NY, USA, 2010. ACM.
4. V. T. Chakaravarthy, H. Gupta, P. Roy, and M. Mohania. Efficiently Linking Text Documents With Relevant Structured Information. VLDB '06, pages 667–678.
5. A. Chandel, P. Nagesh, and S. Sarawagi. Efficient Batch Top-k Search for Dictionary-based Entity Recognition. ICDE '06, page 28, april 2006.
6. H. L. Chieu and H. T. Ng. Named Entity Recognition: A Maximum Entropy Approach Using Global Information. COLING '02, pages 1–7, USA, 2002.
7. X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity Search for Web Services. VLDB '04, pages 372–383. VLDB Endowment, 2004.
8. J. Falkl, R. Laird, T. Carrato, and H. Kreger. IBM Advantage for SOA Governance Standards, Jul 2009. `http://www.ibm.com/developerworks/webservices/library/ws-soagovernanceadv/index.html`.
9. J. Hassell, B. Aleman-Meza, and I. Arpinar. Ontology-Driven Automatic Entity Disambiguation in Unstructured Text. ISWC '06, pages 44–57. 2006.
10. A. Hess and N. Kushmerick. Learning to Attach Semantic Metadata to Web Services. volume 2870 of *ISWC '03*, pages 258–273. 2003.
11. U. Irmak and R. Kraft. A Scalable Machine-Learning Approach for Semi-Structured Named Entity Recognition. WWW '10, pages 461–470, USA. ACM.
12. G. Karypis, E.-H. Han, and V. Kumar. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *Computer*, 32(8):68 –75, aug 1999.
13. P. Malinverno. Service-oriented architecture craves governance, October 2006. `http://www.gartner.com/DisplayDocument?id=488180`.
14. N. Oldham, C. Thomas, A. Sheth, and K. Verma. METEOR-S Web Service Annotation Framework with Machine Learning Classification. SWSWPC '05. 2005.
15. M. Roy, B. Suleiman, D. Schmidt, I. Weber, and B. Benatallah. Using SOA Governance Design Methodologies to Augment Enterprise Service Descriptions. In *CAiSE '11*, pages 566–581. Springer Berlin / Heidelberg, 2011.
16. SAP. Governance for Modeling and Implementing Enterprise Services at SAP, April 2007. `http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/f0763dbc-abd3-2910-4686-ab7adfc8ed92`.
17. E. Saquete, O. Ferrndez, S. Ferrndez, P. Martnez-Barco, and R. Muoz. Combining Automatic Acquisition of Knowledge With Machine Learning Approaches for Multilingual Temporal Recognition and Normalization. *IS'08*, pages 3319 – 3332.
18. E. M. Voorhees. *The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval*. PhD thesis, Ithaca, NY, USA, 1986.
19. B. W. and Watson. A New Algorithm for the Construction of Minimal Acyclic DFAs. *Science of Computer Programming*, 48(2-3):81 – 97, 2003.
20. W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient Approximate Entity Extraction With Edit Distance Constraints. SIGMOD '09, pages 759–770, USA, 2009. ACM.
21. P. Willett. Recent Trends in Hierarchic Document Clustering: A Critical Review. *Information Processing and Management*, 24(5):577 – 597, 1988.
22. I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes : Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, CA, 1999.
23. O. Zamir, O. Etzioni, O. Madani, and R. Karp. Fast and Intuitive Clustering of Web Documents. In *Knowledge Discovery and Data Mining*, pages 287–290, 1997.