Using SOA Governance Design Methodologies to Augment Enterprise Service Descriptions

Marcus Roy[1,2], Basem Suleiman[1], Dennis Schmidt[1],
Ingo Weber[2], and Boualem Benatallah[2]

[1] SAP Research, Sydney NSW 2060, Australia
[2] School of Computer Science and Engineering, UNSW, Sydney NSW 2052, Australia
{m.roy,basem.suleiman,dennis.schmidt}@sap.com,
{m.roy,ingo.weber,boualem}@cse.unsw.edu.au

Abstract. In large-scale SOA development projects, organizations utilize Enterprise Services to implement new composite applications. Such Enterprise Services are commonly developed based on service design methodologies of a SOA Governance process to feasibly deal with a large set of Enterprise Services. However, this usually reduces their understandability and affects the discovery by potential service consumers. In this paper, we first present a way to derive concepts and their relationships from such a service design methodology. Second, we automatically annotate Enterprise Services with these concepts that can be used to facilitate the discovery of Enterprise Services. Based on our prototypical implementation, we evaluated the approach on a set of real Enterprise Service operations provided by SAP. Our evaluation shows a high degree of annotation completeness, accuracy and correctness.

Keywords: SOA Governance, Enterprise Services, Annotation

1 Introduction

Service-oriented Architectures (SOA) allow developers to create flexible and agile composite applications by reusing existing and loosely coupled Web services [5]. Such Web services can be roughly grouped into two categories: public and corporate Web services. On the one hand, there is a large body of public Web services currently available on the Web (cf. seekda[1]). These Web services are typically scattered across various domains (e.g. finance, weather etc.) with heterogeneous service definitions solely based on the service provider's preferences. On the other hand, modern Enterprise Applications, e.g. Enterprise Resource Planing (ERP), are often based on the SOA paradigm enabling organizations using these applications to expose internal data and functionality as (mainly proprietary) Web Services. In this context, Web services are referred to as *Enterprise Service* (ES) [11]. In contrast to public Web services, Enterprise Services are mostly offered through internal[2] and centralized UDDI-like repositories, e.g., SAP's Enterprise Service Repository[3]. Their service design process is typically

[1] http://webservices.seekda.com (28.579 Web Services 11/2010)
[2] Some ESs may also be exposed publicly; both categories are not necessarily mutually exclusive.
[3] ESR: http://www.sdn.sap.com/irj/sdn/nw-esr (>4000 Enterprise Services 11/2010)

standardized and aligned to some kind of business modeling [21], endorsing a close cooperation and coordination of an organization's business and IT departments [27]. In order to manage a great number of ESs, which can easily be in the thousands for real-world enterprise applications [13], some application providers apply governance processes to manage their SOA – summarized under *SOA Governance* – to cover various area of service design, implementation, and provisioning. We herein focus on the design aspect, where SOA Governance often defines best practices to foster high reuse and avoid duplication of Enterprise Services. This includes guidelines to unambiguous naming of interfaces and operations for an increasing number of Enterprise Services. However, the gains in manageability for organizations developing services comes at the price of decreased understandability for service consumers. To demonstrate this point, we refer the reader to the examples of Enterprise Service operation names as shown in Table 1. These have been returned as part of a keyword search for ESs related to "Sales Order" on SAP's ESR. From the perspective of less professional and non SAP-educated developers, these examples of Enterprise Services interfaces may be perceived as (i) long, (ii) technical and (iii) similar sounding. The authors in [3] demonstrated that these characteristics – among others – impose additional difficulties to cognitively understand and locate Enterprise Services.

#	Enterprise Services
1	`SalesOrderItemScheduleLineChangeRequestConfirmation_In`
2	`SalesOrderERPCreditManagementApproveRequestConfirmation_In`
3	`SalesOrderERPItemConditionPropertyByIDQueryResponse_In`

Table 1. Examples of Enterprise Services taken from SAP's ESR

In order to facilitate techniques, e.g. discovery, that can be built on top of Enterprise Services, it is beneficial to augment the description of Enterprise Services as described hereinafter. In most cases, Enterprise Services are defined in terms of a standardized description language, e.g. WSDL[4], that defines what operations can be invoked using which parameters. Description languages like WSDL are, however, not well suited to sufficiently and formally describe the meaning of Enterprise Services. In such cases, semantic languages, e.g. RDF[5], allow to add and relate semantic concepts (e.g. defined by means of ontologies) to parts of the service description. Although semantic approaches have been shown to effectively improve e.g. service discovery and composition, they still require mostly manual effort to create required ontologies and generate related annotations.

In this work, we address the latter point by utilizing service design methodologies from a SOA Governance process to automatically annotate Enterprise Service descriptions. Our approach can be regarded as deriving semantic annotations from ESs to make their discovery easier. We therefore propose a solution to automatically augment the description of Enterprise Services with concepts stemming from such a design methodology. Using the example of an SAP ser-

[4] Web Service Description Language (WSDL) 1.1: `http://www.w3.org/TR/wsdl`
[5] Resource Description Framework: `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`

vice design process, we first identified this service design methodology as well as naming conventions used as part of a SOA Governance process (cf. Figure 1 - Conceptual Layer). Second, we used RDF/S[6] to describe and represent this

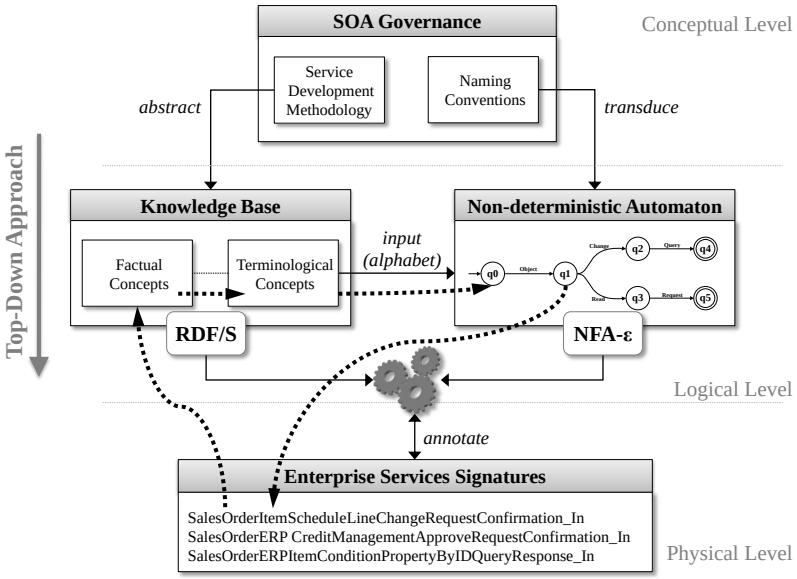

Fig. 1. Architectural overview of our approach to utilize SOA governance

service development methodology, abstractly. We formally defined this representational schema as a hierarchy of terminological and factual concepts, referred to as service knowledge base (cf. Figure 1 - Logical Layer). Concepts in that knowledge base – among others – are used to define signatures of Enterprise Services. We then use non-deterministic automata to formally define a language of Enterprise Service signatures derived from available naming conventions (cf. Figure 1 - Logical Layer). We refer to the knowledge base and automaton as service annotation framework. Third, we applied our framework to the example of SAP to automatically annotate Enterprise Services (cf. Figure 1 - Physical Layer). That is, an Enterprise Service signature that is accepted by the automaton is a valid concatenation of words (i.e. factual concepts) that relate to terminological concepts in the service knowledge base. The set of detected concepts is then used to annotate the respective Enterprise Service. The corresponding steps are illustrated as dashed lines in Figure 1. Our prototypical implementation shows that a majority of available Enterprise Services can be annotated automatically. An evaluation of generated annotations in fact demonstrates a high degree of completeness, accuracy and correctness.

In the remainder, we explain an example of SOA Governance in Section 2. We then describe the service annotation framework in Section 3. In Section 4, we present a solution and evaluate generated annotations in Section 5. Finally, we refer to related work in Section 6 and provide a conclusion in Section 7.

[6] RDF Schema: `http://www.w3.org/TR/rdf-schema`

2 SOA Governance

Organizations use SOA Governance to better manage their SOA development. This has been ascertained by [16], predicting a considerably high risk of failures for midsize to large SOA projects of more than 50 Web Services without any applied SOA governance mechanism. Large companies such as SAP, IBM and Oracle employ SOA Governance mechanisms to ensure a consistent, effective and business aligned development of SOA-based applications. We refer to SOA Governance as policies to make consistent decisions on how to build usable and long-living services. SOA Governance, similar to IT Governance, typically covers multiple phases of a service life-cycle, e.g. service planning, design, definition, implementation etc. [10]. In terms of service design, developers are guided in their task to create interfaces of future Enterprise Services. Governance applied during the design phase typically encompasses guidelines and best practices to effectively create services that are (ideally) mutually exclusive and exhaustive regarding coverage of functionality. It also creates a common agreement and semantic alignment of concepts used during the service development. For instance "Sales Order" is a business entity defining a contractual order that is commonly understood by developers, customers and partners across corporate boundaries.

2.1 SOA Governance - A Service Design Example

As a motivating example, we describe one possible way to abstract and utilize information used as part of SAP's service development methodology for creating Enterprise Services. Specifically the definition of Enterprise Service signatures can be quite versatile, encompassing the use of multiple (i) concepts arranged by some kind of (ii) naming convention [2, 21]. We herein refer to concepts as *terminological concepts* and to its instance data as *factual concepts*. The schema in Figure 2 shows an example of such concepts as part of our definition of a *knowledge base*. Particularly, the schema illustrates – but is not restricted to – the application of two main terminological concepts: a domain-specific *data model* and service development *pattern*. We used RDF/S to describe the hierarchy of both types of concepts in a single schema focusing only on a minimal example of terminological concepts, i.e. data model and pattern, and naming conventions used to describe Enterprise Services. We refer to the Enterprise Service (S1) as an end-to-end example to illustrate used design principles as well as our approach:

`SalesOrderItemScheduleLineChangeRequestConfirmation_In` (S1)

Signature (S1) defines an incoming request operation (invoked by a seller) to change the schedule line of an item contained in an existing sales order of a specific customer. From this signature, we can recognize factual concepts that have been modeled as part of our schema. On the one hand, the terms "Sales Order", "Item" and "Schedule Line" refer to factual concepts belonging to terminological concepts listed under data model (cf. Fig. 2). On the other hand, the terms "Change", "Request" and "In" represent factual concepts of a particular service development pattern, also a terminological concept. In a final step, these

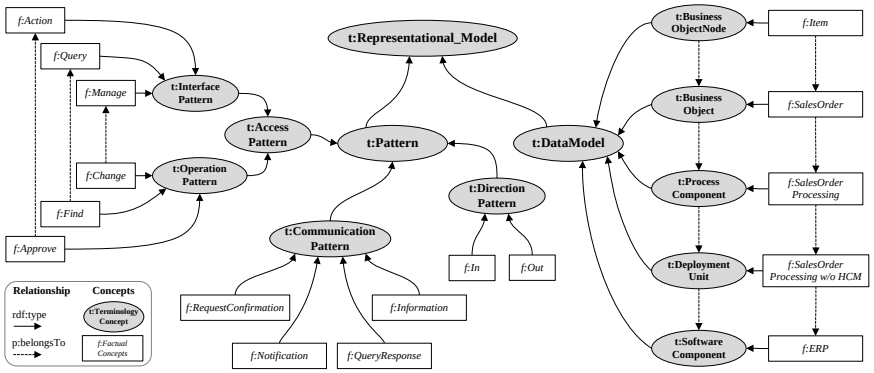

Fig. 2. RDF Schema of the knowledge base showing terminological and factual concepts

terms are arranged according to a specific order that is defined by a set of naming conventions. Next, we explain terminological concepts of the data model and pattern and show how they are utilized to create Enterprise Service signatures using naming conventions.

A domain-specific data model Typically, Enterprise Services are built on key entities of a domain-specific data model. For instance, signature (S1) is specified according to the factual concept "Sales Order", "Item" and "Schedule Line" as inferred from the data model defined as part of the knowledge base (cf. Fig. 2). This model represents an abstract and trimmed-down version of an existing information model used by SAP. Such an information model describes, in more detail, the relationship of business-related entities used by an organization to realize internal operations. Each terminological concepts has factual concepts associated with it. Applying this to our example, the terminological concept "Business Object Node" (BO Node) belongs to "Business Object" (BO), which implies that the factual concept "Item" belongs to "Sales Order": `SalesOrderItemScheduleLineChangeRequestConfirmation_In`

Service development patterns Apart from the data model, the service knowledge base also describes development patterns recurrently used by developers to uniformly define Enterprise Services. These patterns can further be separated into access, communication and direction pattern (cf. Fig. 2). Firstly, the access pattern specifies predefined ways on how to access objects in the data model, e.g. "create" or "change". Secondly, the communication pattern describes the type of interaction to be e.g. "request-confirmation" to define a request operation that causes some action and a confirmation message being returned. Thirdly, the direction pattern describes a service operation to be either "inbound" (e.g. incoming request to create a sales order) or "outbound" (e.g. outgoing credit card authorization request). We derived these patterns directly from available SOA Governance information. Referring to (S1), the signature employs factual concepts "change", "request-confirmation" and "in" belonging to terminological concepts access, communication and direction pattern respectively: `SalesOrderItemScheduleLineChangeRequestConfirmation_In`

Naming Conventions The examples (N1)-(N3) below illustrate that naming conventions already come in a pre-structured way (cf. [21]) and are used to guide the creation of Enterprise Service signatures. The showcased syntax is defined on the basis of non-terminal symbols, i.e. terminological concepts, and terminal symbols, i.e. factual concepts.

$$\texttt{NC}_1 = \texttt{<BO>(<BO Node>)}^* \tag{N1}$$

$$\texttt{NC}_2 = \texttt{("Change"|"Create"|"Cancel"|"Read")} \tag{N2}$$

$$\texttt{NC}_3 = \texttt{<CommunicationPattern>} \tag{N3}$$

In terms of signature definition, (N1) for instance describes that any factual concept related to a BO Node has to be positioned after the factual concept of a particular BO. Finally, to receive a single (possibly incomplete) building rule, we consolidated naming conventions (N1), (N2) and (N3) based on documentation that outlines how to combine them. We further substituted the access pattern for the terminal words in (N2). As a result, the building rule (B1) represents one possible way of (partially) describing our Enterprise Service signature (S1):

$$\begin{aligned}\texttt{BR} &= \texttt{<BO><BO Node><AccessPattern><CommunicationPattern>} \\ &= \underline{\texttt{SalesOrderItem}}\texttt{ScheduleLine}\underline{\texttt{ChangeRequestConfirmation\_In}}\end{aligned} \tag{B1}$$

3 Automated Annotation Framework

In this section, we formally describe the two components used in our framework to automatically annotate Enterprise Services, i.e. the service knowledge base and the non-deterministic automaton. These components are based on our modeling of SOA Governance as described in the previous Section. Therefore, we first define the service knowledge base as an abstract model and data that represents a given service design methodology (cf. data model and pattern in Section 2.1). Second, we define a non-deterministic automaton describing a formal language of Enterprise Service signatures. To illustrate this procedure, we will use the naming conventions from Section 2.1.

3.1 Service Knowledge Base

In the following, we formally define the service knowledge base as $\mathcal{KB} = \langle D, F \rangle$. We refer to D as a set of terminological concepts describing an abstract representation of a service development methodology. With F, we describe a set of factual concepts that are mapped to related terminological concepts in D.

Definition 1 (Methodology Representation D). *We define the abstract representation D as a directed graph $D = (T, R, E)$ with T representing a set of terminological concepts $T = \{t_1, \ldots, t_n\}$, R denoting a set of relationships $R = \{r_1, \ldots, r_m\}$ and E a set of directed edges between two terminological concepts belonging to a specific relationship such that $E = \{e_1, \ldots, e_k\}$ with $e_i = (t_o, r_y, t_p)$, $0 \leq i \leq k$, $0 \leq o, p \leq n$, $0 \leq y \leq m$.*

Example 1 (Methodology Representation D). We use the example of terminological concepts as described in Section 2.1 to define the conceptual part $D = (T, R, E)$ of the service knowledge base $\mathcal{KB} = \langle D, F \rangle$. As such, the representation D describes a child-relationship of the concept "Business Object Node" to the concept "Business Object".

$T := \{t_1, t_2\} = \{\texttt{Business Object}, \texttt{Business Object Node}\}$,
$R := \{r_1\} = \{\texttt{containsBON}\}$, $E := \{e_1\}$ with
$e_1 := (t_1, r_1, t_2) = (\texttt{Business Object}, \texttt{containsBON}, \texttt{Business Object Node})$

Definition 2 (Factual Concepts F). *We define F as a set of factual concepts $F = \{f_1, f_2, \ldots, f_m\}$. We further define a mapping $\Phi : F \to T$, $\Phi(f) = t$ for $f \in F, t \in T$, such that $\forall f \in F : \exists t \in T : \Phi(f) = t$. Furthermore, for each $t \in T$ we denote the (possibly empty) subset $F_t \subset F$ such that $\forall f \in F_t : \Phi(f) = t$. Obviously these subsets are distinct for different t, i.e. $\forall t_i, t_j \in T : t_i \neq t_j \rightarrow F_{t_i} \cap F_{t_j} = \emptyset$*

Example 2 (Factual Concepts F). Referring to the examples of factual concepts as shown in Section 2.1, we use F to represent a set of factual concepts, i.e. "Sales Order", "Purchase Order" and "Item". We further have distinct subsets F_{t_1} and F_{t_2} of F, whereas "Sales Order" and "Purchase Order" represent F_{t_1} and "Item" forms F_{t_2}. The mapping Φ describes the relationship of factual concepts in F_{t_1} and F_{t_2} to T, which practically relates "Sales Order" and "Purchase Order" to "Business Object" and "Item" to "Business Object Node".

$F := \{f_1, f_2, f_3\} = \{\texttt{Sales Order}, \texttt{Purchase Order}, \texttt{Item}\}$,
$\Phi(f_1) := t_1 \rightarrow \Phi(\texttt{Sales Order}) = \texttt{Business Object}$
$\Phi(f_2) := t_1 \rightarrow \Phi(\texttt{Purchase Order}) = \texttt{Business Object}$
$\Phi(f_3) := t_2 \rightarrow \Phi(\texttt{Item}) = \texttt{Business Object Node}$
$F_{t_1} := \{f_1, f_2\} = \{\texttt{Sales Order}, \texttt{Purchase Order}\}$, $F_{t_2} := \{f_3\} = \{\texttt{Item}\}$

3.2 Service Signature Automaton

In this section, we use the notation of a non-deterministic finite automaton (NFA) with ε-moves to formally define a language of accepted Enterprise Services signatures. We use previously defined terminological concepts T (Def. 1) as the set of input symbols (i.e. alphabet) to initiate state changes. A path through this automaton, i.e. a finite sequence of connected states, ending in a final state represents a concatenation of terminological concepts that defines a language of Enterprise Service signatures.

Furthermore, we decided for an NFA-ε and against a DFA (deterministic finite automaton) for the following reason. Although we expect SOA governance-compliant Enterprise Services to correctly employ naming conventions, we cannot assume them to be exhaustive to completely describe any Enterprise Services signature. This means that the automaton should be able to ignore parts of the signature that are unknown, i.e. not defined by a naming rule or where concepts are not recognized. For this, we included empty-word transitions (ε-moves).

Definition 3 (Automaton A). *We define the NFA-ε as $A = (Q, T, M, q_0, Z)$ with Q denoting a finite set of states, T used as input symbols, $M : Q \times (T \cup \{\varepsilon\}) \to P(Q)$ as the transition function (including ε-moves) to a powerset of Q, $q_0 \in Q$ representing the start state and $Z \subseteq Q$ denoting a (possibly empty) set of final states. We further define the powerset of a particular state $P(\{q\})$, $q \in Q$ as the set of states that can be reached from q with input $t \in T$ and ε such that $P(\{q\}) = \{p \in Q : q \xrightarrow{t,\varepsilon} p\}$ (ε-closure). The powerset of all states is defined as a union $P(Q) = \bigcup_{q \in Q} P(\{q\})$.*

Example 3 (NFA-ε). In Figure 3, we depicted an example of an automaton consisting of nine states $Q = \{q_0, \ldots, q_8\}$, an alphabet of nine symbols $T = \{t_1, \ldots, t_8, \varepsilon\}$, two accepting states $Z = \{q_7, q_8\}$ and a set of transitions M represented as edges in Figure 3. We further refer to the following examples of Enterprise Service signatures (S1) and (S2) that are accepted by this automaton using the set of transitions M_{S1} and M_{S2}. Moreover, Enterprise Service signature (S2) illustrates the need for ε-moves.

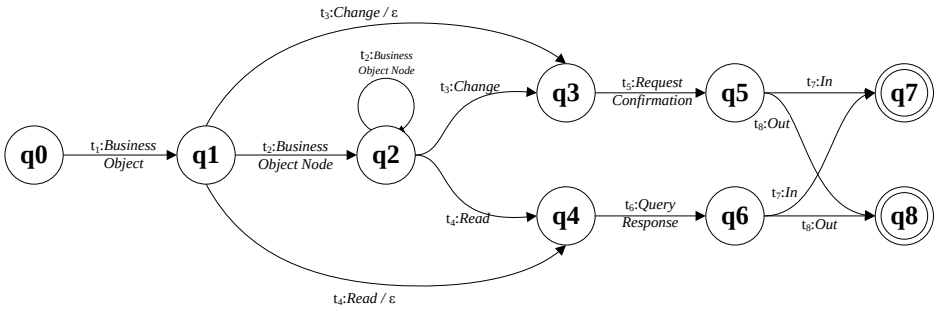

Fig. 3. Example of a NFA-ε accepting e.g. Enterprise Service signature (S1) and (S2)

(S1) `SalesOrderItemScheduleLineChangeRequestConfirmation_In`

$$M_{S1} = \{q_0 \xrightarrow{\Phi(\texttt{Sales Order})} q_1, q_1 \xrightarrow{\Phi(\texttt{Item})} q_2, q_2 \xrightarrow{\Phi(\texttt{Schedule Line})} q_2, q_2 \xrightarrow{\Phi(\texttt{Change})} q_3, q_3 \xrightarrow{\Phi(\texttt{Request Confirmation})} q_5, q_5 \xrightarrow{\Phi(\texttt{In})} q_7\}$$

(S2) `SalesOrderCRMItemSimpleByIDQueryResponse_In`

$$M_{S2} = \{q_0 \xrightarrow{\Phi(\texttt{Sales Order})} q_1, q_1 \xrightarrow{\varepsilon} q_4, q_4 \xrightarrow{\Phi(\texttt{Query Response})} q_6, q_6 \xrightarrow{\Phi(\texttt{In})} q_7\}$$

4 Automated Annotation Solution

In this section, we briefly describe the construction of automata and an accepting algorithm used to detect concepts for the service annotation.

4.1 Automaton Construction

Automata, as defined in the previous section (cf. Def. 3), are generated from a set of rules that represent naming conventions. As referred to in Section 2.1,

these rules consist of the following two parts. Firstly, there are non-terminal and terminal elements, e.g., `<BO>` and `SalesOrder` respectively, representing conditions for automata transitions as shown in (N1) - (N3) in Section 2.1. Each rule is transformed into a regular expression, which is abstractly defined using terminological terms $t \in T$, e.g. `/<BO>/`. Subsequently we replace the terminological concepts with the respective set of factual concepts $f \in F$ from the knowledge base – for instance `/<BO>/` $\rightarrow$ `/(SalesOrder|PurchaseOrder|...)/`.

Secondly, concatenation instructions define the automaton's structure as shown in (B1) in Section 2.1. These instructions specify the actual states. For each of these states, we assign potential input transitions, from the just created transition set, that need to be satisfied to enter this state. Furthermore, these rules define the states that are subsequently reachable.

Based on these constructed automata, Enterprise Service signatures are received as input to start the annotation procedure as described hereinafter.

4.2 Annotation Procedure

The actual annotation of Enterprise Services is realized by Algorithm 1, which accepts Enterprise Service signatures by detecting used concept. This accepting algorithm works recursively, starting with state q_0. Each state checks its incoming state transition, i.e. compares its own regular expression against the input provided. If the input transition is satisfied (line 1), i.e. a valid concept was found, the matched part is stored (line 2) and removed from the beginning of the input and passed on to its subsequent states (line 5). In case of an empty-word transition, obviously the same input is passed on to its subsequent states. When the base case has been reached in form of an accepting state (line 3), it returns its matched concept to its predecessors (line 13). For each of its preceding states,

Algorithm 1 Annotate Enterprise Service: $annotate(q, i)$

Input: $q \in Q; i \in F^*$
Output: $A \subseteq TF; TF = \{(t, f)\}, t \in T, f \in F$

1: **if** $\exists w, w', q' : i = ww' \wedge w, w' \in (F \cup \{\epsilon\})^* \wedge q \xrightarrow{\Phi(w)} P(\{q\}) \wedge q' \in P(\{q\})$ **then**
2: $A \leftarrow \{(\Phi(w), w)\}$
3: **if** $q' \notin Z$ **then**
4: **for** $p \in P(\{q\})$ **do**
5: $S' \leftarrow annotate(p, w')$
6: **if** $|S'| > |S|$ **or** S is undefined **then** {choose only the best path}
7: $S \leftarrow S'$
8: **end if**
9: **end for**
10: $A \leftarrow A \cup S$ {S undefined $\rightarrow$ A undefined}
11: **end if**
12: **end if**
13: **return** A {undefined A indicates a failed path}

Call: annotate(q_0, SalesOrderItemScheduleLineChangeRequestConfirmation_In)

a set of matched concepts, i.e. their own plus previously matched concepts (line 10), is returned to their predecessors and so forth. The choice on what match results are returned depends on the quality of matched concepts. Therefore, the result with the largest number of different concepts covering most of the input is chosen (line 6 and 7). Eventually, the recursive algorithm terminates, detecting a maximum number of concepts used to annotate Enterprise Services.

We implemented this algorithm in Java with roughly 2000 lines of code. The knowledge base and generated annotations are stored as RDF triples. The Sesame framework in version 2.3.2[7] has been used for RDF storage, querying, and inferencing.

5 Automated Annotation Evaluation

In this section, we evaluate our automated annotation approach using our prototypical implementation (cf. Section 4) based on a set of Enterprise Services. We then analyze the annotation results in terms of *completeness*, *accuracy* and *correctness*, which we define throughout the section. These criteria allow a direct evaluation of our annotation approach; a more exhaustive user study will be conducted once we devised a fully-fledged search.

5.1 Evaluation Environment

We conducted our evaluation using 1654 Enterprise Service signatures taken from SAP's ARIS Designer from various SAP applications, such as ERP, CRM, etc. These services are from the group of so-called A2X (Application-to-Unknown) ESs, as they have a coherent naming scheme. Based on our representational model, i.e. terminological concepts and their relationships, we automatically extracted the corresponding instance data, i.e. the factual concepts, from semi-structured sources to populate the service knowledge base. Examples of these sources are Web pages from SAP's Enterprise Service Workplace that provide documentation for each Enterprise Service as well as internal documents (Excel sheets and the like) exported from SAP's ARIS platform. To extract and store this information as RDF triples, we developed extractors for each source of information (i.e. HTML/XML/Excel extractor). As a next step, we used documents describing naming conventions [21] to define the corresponding automaton as described in Definition 3. Finally, we used the 1654 Enterprise Services as the input to the algorithm as described in the previous section. The resulting set of detected concepts has been stored as annotations in form of RDF triples referencing the original Enterprise Service. This is the basis of the analysis below.

5.2 Annotation Completeness

The annotation completeness represents the number of Enterprise Services that have been partially or fully annotated. For this, we calculated the expected

[7] http://www.openrdf.org/

maximal number of annotations for each Enterprise Service operation by taking advantage of their Camel Case notation. We refer to the annotation accuracy as the ratio of actual number of generated annotations compared to the expected number of annotations. To determine the annotation completeness, we only considered Enterprise Services with an accuracy greater than null. As a result, we achieved an overall annotation completeness of 1583 out of 1654 Enterprise Services, which is equivalent to 95.7%. The missing 4.3% stem from Enterprise Service signatures that did not comply with existing naming conventions.

5.3 Annotation Accuracy

In this part of the evaluation, we only considered the 1583 fully or partially annotated Enterprise Services from above. To determine the accuracy of annotations, we grouped them into categories of 100% to 40% annotation accuracy (using a 10% scale). In terms of annotation accuracy, we refer to the ratio of actual vs. expected annotations from the previous section. We set the lower margin to 40% based on the lowest accuracy of all 1583 annotated Enterprise Services. For that level of accuracy, we only found four Enterprise Services. In fact, less than 1% of Enterprise Services have been annotated with less than 50% accuracy. On the other hand, the majority of Enterprise Services, i.e. 73.0%, have been fully annotated as illustrated in Figure 4. For an annotation accuracy of 80% or more, the percentage of annotated Enterprise Services increases to 91.4%. The whole procedure on the entire data set took less than 5 minutes on an Intel(R) Core(TM)2 Duo Processor T7300 machine @ 2GHz CPU and 3GB of RAM. These numbers lead to two observations: (i) the naming conventions were largely followed in the tested sample of ESs, and (ii) our approach delivered an effective annotation.

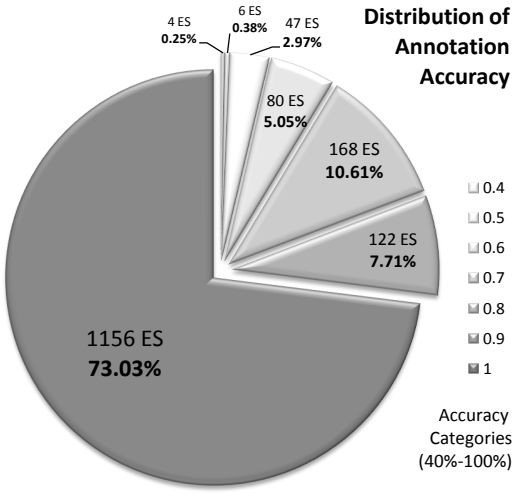

Fig. 4. Result Categories of the Annotation Accuracy Experiment

5.4 Annotation Correctness

To the best of our knowledge, there is no obvious solution to automatically validate the correctness of any generated annotation; the baseline therefore is manual verification. Therefore, we first selected a 10% sample of the completely annotated Enterprise Services to evaluate their correctness. Half of these services were strategically selected by a domain expert to cover various applications as well as a variety of design concepts and naming conventions. The other half was randomly selected to avoid any biased decision regarding the selection of concepts. In a second step, an independent expert in SOA Governance has been briefed on the workings of the service annotation approach. He then manually performed the annotation based on the technical names of these 10% sample Enterprise Services. In a final step, the expert's manual annotations have been analyzed and compared to the automatically generated annotations by one of our developers. As a result, we confirmed that approximately 94% of automated annotations match the manual annotations in terms of number of annotations and annotated concepts. We further investigated the 6% range of annotation mismatches, which we separated into two categories: governance violation and missing record. The former category describes Enterprise Services that, for whatever reason, did not fully comply with the SOA Governance design methodology. The latter denotes that the specific concept is missing in the service knowledge base and has been wrongly annotated as a different concept. While mismatches resulting from governance violations are out of our control, mismatches stemming from missing records can be fixed by extending the knowledge base. Either way, we believe that mismatches of less than 6% can be considered acceptable.

6 Related Work

Before we examine related work on proposed annotation techniques, we first outline some fundamental approaches that have been proposed to add additional knowledge (metadata) to the technical description of services, e.g WSDL. For instance, linguistic approaches using clustering techniques – as done in the Woogle search engine [8] – aim to derive meaningful concepts from the technical description of Web Services using WSDL. Although it effectively improves the search, the approach lacks certainty about generated concepts making an automatic solution less feasible. In contrast, semantic approaches [4, 25, 23, 26, 1, 18] are based on an explicit modeling of ontologies which capture a specific domain knowledge in a generic way using concepts that are commonly agreed and understood [9]. As a consequence, machines become more able to interpret data and documents that are annotated with concepts from ontologies requiring less or no human intervention. In general, research communities distinguish between two types of metadata, i.e. ontological concepts and annotations encoding references to concepts.

In terms of ontological concepts, we described what can be seen as a rather closed ontology derived from a given methodology. Such an ontology can be considered nearly complete with respect to the universe of discussion. However,

we did not focus on the semi-automatic creation of ontologies [15, 6, 12] such as generating ontologies from natural language text or automatic reuse of existing ontologies, e.g. based on the context of a document [22]. In fact, reusing public ontologies, e.g. TAP [7] or others, are less applicable in context of our work as design methodologies in an enterprise context rather describe proprietary concepts (e.g. data model) that cannot be necessarily expressed with public or community-contributed [24] ontologies.

In terms of annotations, existing approaches can be first categorized into degrees of automation, i.e. fully-automated, semi-automated [9, 15] or manual [14, 19]. Second, the annotation level of detail depends on the type of representation ranging from e.g. entire service descriptions, operation or input/ouptut parameters, non-functional requirements etc. Third, annotations can be used for different fields of applications, e.g. human or automatic service discovery [17], invocation, composition [22] etc. In this work, we proposed a fully-automated approach to annotate a large set of Enterprise Services based on their signatures, i.e. interface and operation names. Ultimately, we currently see the main purpose of the service annotation approach in improving the service discovery specifically for business users and non-professional developers. In this context, using automata can more precisely resolve disambiguities by determining correct concepts based on their expected position within the respective signature. As a result, we can increase the accuracy of generated annotations compared to other approaches, e.g. using similarity functions [7] (SemTag), natural phrase processing [9] (via SMES) or detecting association of concepts by confidence level [15]. Note, this is particularly feasible as we only focus on a small portion of explicitly defined text, i.e. Enterprise Service signatures, rather than a large body of text, i.e. Web pages.

For reasons of simplicity, we used RDF(/S) over OWL-S or WSMO to store our ontology and annotations. We consider RDF(/S) sufficient for our purpose to represent additional knowledge as we do not require sophisticated logical reasoning. The results of the annotation could, e.g., be stored in SAWSDL [25, 23] format, where annotations are directly added to WSDL. However, at this point we are not entitled to change Enterprise Service descriptions, and decided to keep annotations independent from any specific Web Service standard.

7 Conclusion and Future Work

In this paper we presented an approach for the automatic annotation of Enterprise Services, based on a SOA Governance design methodology. We described a concrete methodology used at SAP, but presented a generic and formal model for capturing the structure of SOA Governance design methodologies. The model consists of terminological concepts and factual concepts, and automata for capturing naming conventions built from these concepts. Naming rules are specified using a (typically very small) set of terminological concepts; from those we construct a consolidated automaton and populate it with the respective factual concepts. Using the detailed automaton, we can automatically annotate service names that (at least partially) adhere to the naming conventions.

We evaluated the work on a set of more than 1500 Enterprise Services from SAP, and obtained highly encouraging results: more than 90% of the services could be annotated with more than 80% correctness. This was largely verified in a small experiment with an independent expert. We observed that some of the mismatches came from ESs that did not adhere to naming conventions. As such, our approach can also be used to check adherence to naming conventions, and thus, improve the management of SOA Governance. In terms of the annotation procedure, this is an one-off operation that only needs to be executed when the concepts or the service names change. All the above services were annotated in a matter of minutes. Hence, performance is unlikely to be a problem.

In an earlier instance of this work [20], we used a strongly simplified model, yielding significantly lower accuracy: only parts of the data model and patterns could be annotated. However, we there also showed how these annotations can be used in discovery of Enterprise Services for business users and developers who are unfamiliar with the set of services. As such, we can use our approach to facilitate an effective search by further reaching out into areas of automatic query extension and query suggestion.

Future work will build on the presented approach as follows. Firstly, we will attempt to evaluate the applicability of the approach on services from other sources; there is, however, a high risk that we might not be able to obtain detailed access to naming rules. Secondly, we will investigate improved search algorithms and other application scenarios making use of the produced annotations.

References

1. R. Akkiraju, J. Farell, J. A. Miller, M. Nagarajan, A. Sheth, and K. Verma. Web service semantics – WSDL-S. In *W3C Workshop on Frameworks for Semantic in Web Services*, 2005.
2. D. J. Artus. SOA Realization: Service Design Principles, February 2006. `http://www.ibm.com/developerworks/webservices/library/ws-soa-design/`.
3. J. Beaton, S. Y. Jeong, Y. Xie, J. Stylos, and B. A. Myers. Usability Challenges for Enterprise Service-oriented Architecture apis. In *VLHCC '08: Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 193–196, Washington, DC, USA, 2008. IEEE Computer Society.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
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. De Silva and L. Jayaratne. Wikionto: A System for Semi-automatic Extraction and Modeling of Ontologies Using Wikipedia XML Corpus. In *Semantic Computing, 2009. ICSC '09. IEEE International Conference on*, pages 571 –576, 2009.
7. S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, K. S. Mccurley, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. A Case for Automated Large Scale Semantic Annotations. *Journal of Web Semantics*, 2003.
8. X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity Search for Web Services. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 372–383. VLDB Endowment, 2004.

9. M. Erdmann, A. Maedche, H.-P. Schnurr, and S. Staab. From Manual to Semi-automatic Semantic Annotation About Ontology-based Text Annotation Tools. In *Proc. of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*, Luxembourg, August 2000.
10. T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall Professional Technical Reference, 2005.
11. V. Haentjes. SOA Made Easy with SAP, February 2010. `http://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/903aa937-03f2-2c10-968e-8e7d649cd352`.
12. F. V. Harmelen and D. Fensel. Practical Knowledge Representation for the Web. In *In Proc. of the 2000 Description Logic Workshop (DL 2000*, pages 89–97, 1999.
13. J. Hoffmann, I. Weber, and F. M. Kraft. SAP Speaks PDDL. In *AAAI*, 2010.
14. P. Kungas and M. Dumas. Cost-Effective Semantic Annotation of XML Schemas and Web Service interfaces. In *SCC '09. IEEE International Conference on Services Computing*, pages 372–379, Sept. 2009.
15. A. Maedche and S. Staab. Semi-automatic Engineering of Ontologies from Text. In *Proc. of 12th Int. Conf. on Software and Knowledge Eng.*, Chicago, IL, 2000.
16. P. Malinverno. Service-Oriented Architecture Craves Governance, October 2006. `http://www.gartner.com/DisplayDocument?id=488180`.
17. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In *ISWC'02*, 2002.
18. A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma. Meteor-S Web Service Annotation Framework. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 553–562, New York, NY, USA, 2004. ACM.
19. J. Rao, D. Dimitrov, P. Hofmann, and N. Sadeh. A Mixed Initiative Approach to Semantic Web Service Discovery and Composition: SAP's Guided Procedures Framework. In *ICWS '06: Proc. of the IEEE Int. Conf. on Web Services*, pages 401–410, Washington, DC, USA, 2006. IEEE Computer Society.
20. M. Roy, B. Suleiman, and I. Weber. Facilitating Enterprise Service Discovery for Non-Technical Business Users. In *ICSOC'10: 6th International Workshop on Engineering Service-Oriented Application*, 2010. (to be published).
21. SAP AG. 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`.
22. A. Segev and E. Toch. Context-Based Matching and Ranking of Web Services for Composition. *IEEE Transactions on Services Computing*, 2009.
23. K. Sivashanmujgam, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Services Standards. In *Int. Conference on Web Services (ICWS'03)*, June 2003.
24. S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, and Y. Sure. Semantic Community Web Portals. In *Proc. of the 9th Int. WWW conference on Computer Networks*, pages 473–491, Amsterdam, The Netherlands, 2000. North-Holland Publishing Co.
25. K. Verma and A. Sheth. Semantically Annotating a Web Service. *IEEE Internet Computing*, 11(2):83–85, 2007.
26. T. Vitvar, A. Mocan, M. Kerrigan, M. Zaremba, M. Zaremba, M. Moran, E. Cimpian, T. Haselwanter, and D. Fensel. Semantically-enabled Service Oriented Architecture : Concepts, Technology and Application. *Service Oriented Computing and Applications*, 1(2):129–154–154, June 2007.
27. B. Woolf. Introduction to SOA Governance, July 2007. `http://www.ibm.com/developerworks/library/ar-servgov/`.