Behavior Classes for Specification and Search of Complex Services and Processes

Martin Junghans, Sudhir Agarwal, and Rudi Studer
Karlsruhe Institute of Technology (KIT)
Institutes AIFB and KSRI
Englerstr. 11, 76131 Karlsruhe, Germany
Email: firstname.lastname@kit.edu

***Abstract*—In the Web there are a large number of (business) services with complex behavior, such as e-commerce Web sites that require multiple interactions with the user, as well as an increasing number of Web automation scripts to coordinate the execution of multiple complex services. However, while there are a quite a few search techniques for atomic Web services, search techniques for complex services are still rare and only foundational. In this paper, we present *behavior classes* that have formal semantics as well as human comprehensible names in order to foster usability of specification of constraints, and efficiency of search for complex services and processes. Our approach enables automatic methods for (i) assigning behavior classes to complex behavior descriptions, (ii) checking consistency of such a classification, and (iii) computing behavior class hierarchies. Furthermore, human comprehensible names for the behavior classes increase usability by allowing for shorter service descriptions and requests. Our evaluation results show that a behavior class hierarchy can be exploited as an indexing structure to gain performance of search.**

I. Introduction

In the Web, a large number of services are offered in form of Web sites that have complex behavior (multiple interactions with the user that may even depend on the user input at previous interactions of the same process run etc.). Furthermore, there are often multiple Web sites offering the same or similar functionalities and information. In order to gain a broader overview of the desired product, information or functionality, a user often needs to follow several paths on various Web sites by providing right inputs at right time, and accepting the (intermediary) outputs.

Different Web sites offer different granularities of information and functionalities and have heterogeneous navigation paths. Logical dependencies between the information and functionalities provided by different Web sites effect the order they are executed by a user. This makes it difficult for end users to coordinate the execution of various Web sites, and aggregate the information gathered from them. Web automation scripts, originally developed for the purpose of testing Web sites by developers, are turning out to be promising for end users as well, since they can automate this tedious process to a large extent. However, finding and composing Web automation remains very difficult, since it requires a lot of manual effort due to the huge gap between the user requirements and the functionality offered by existing search techniques. In order to find and compose complex executable models, users need to be able to search by constraining the temporal behavior of the models as well as the information they require and deliver at various stages during their execution. However, existing search techniques support such search functionality only for atomic Web services. Search for Web sites and Web automation scripts is based upon syntactic matching of keywords with the content at the surface Web (as opposed to Deep Web), or with manually added tags of the script respectively. Even though the tags could hint at the functionality of a script, tagging requires manual effort and is often faulty [1]. In order to equip users with the power of finding, creatively combining, and executing Web sites for emerging more and more sophisticated use cases, a process-oriented view on Web sites is required. Even if the syntactic keyword-based approaches were extended to support the process-oriented view, they would still be restricted to support end users only. If an execution engine encounters problems like a failing service during runtime, it is often desired that it finds and composes an alternative solution automatically, and resumes the execution. However, the requirements for an alternative are known to system in form of structural constraints that have a formal semantics in the first place, and not as ambiguous natural language keywords. Formalisms available for modeling complex distributed processes and automatic reasoning about them including our previous work are neither easy to use for end users, nor do the exhibit acceptable performance and scalability for practical purposes.

In this paper, we propose a way out of this problem by introducing the notion of *behavior classes* that have human comprehensible names as well as formal definitions and advance our work on the classification of atomic services [2]. The formalisms developed in our previous works [3] for semantically describing the observable behavior, and for specifying behavior constraints resp. are briefly presented in Section II. In Section III, we extend our previous approach by allowing enrichment of observable behavior description with not directly observable temporal properties. In Section IV, we present the syntax and semantics of behavior classes, and show how hierarchies of behavior classes can be built and kept consistent. We then show the added value of behavior classes in modeling a complex behavior. In Section V, we show how behavior classes can be used as predefined query fragments within a

search request, as well as exploited as an indexing structure to achieve faster query answering. Section VI presents details of the prototypical implementation of our approach as well as performance evaluation results. We draw conclusions in Section VIII, after discussing and relating our work with existing approaches in Section VII.

II. Preliminaries

Process formalisms allow for the description of observable behavior of a service or a process by providing constructs for modeling data and control flow. Concrete actions with concrete parameter values occurring in process runs are aggregated to variables. Behavioral constraints express desired properties of a behavior. Model-checking techniques that check automatically whether a given behavior description satisfies given set of constraints serve as a fundamental reasoning task for searching for complex services and processes.

A. Description of Observable Behavior

We view the observable behavior of a complex service or a process as a labeled transition system (LTS). An LTS $L = (S, T, \rightarrow)$ is defined by a finite set S of states, a set T of transition labels, and transitions $\rightarrow$: $S \times T \times S$ between states. The knowledge of a state is described by the facts that hold in the state. The transitions represent input, output, or computational activities.

We use a combination of the $\mathcal{SHOIN}(\mathbf{D})$ description logic (DL) ABox axioms [4] and the π-calculus process algebra [5] to describe the executable behavior of complex services and processes that can be interpreted as an LTS [3], [6]. The language comprises constructs for input, output, and computational activities, parallelism, conditions, invocation of services and processes, and the specification of communication channels. In a $\mathcal{SHOIN}(\mathbf{D})$ ABox we describe data objects and their relationships within process expressions. Changes that occur during the execution are modeled with ABox change axioms like adding or deleting DL individuals or relations among the individuals. Transition labels store information about performed changes caused by input, output, or computational activities.

Example[1] A train booking Web site implements a complex behavior described by a process. The user interactions implemented by forms are modeled by input activities, e.g., asking for date, location, and preferences of a train connection. Fetching available connections for the given requirements is modeled by a computational activity. As an effect, the connections are added as individuals related to the parameters of the previous input activity into the state after the computation.

B. Specification of Behavior Constraints

The specification of the complex behavior of a service or a process is achieved by the following syntax:

$$\Psi := \Psi \wedge_\mu \Psi \mid \neg_\mu \Psi \mid \mu X.\Psi(X) \mid \langle A \rangle \Psi \mid P \mid \top \mid \bot$$

[1]We show the complete example in http://people.aifb.kit.edu/mju/suprime. Domain ontologies originate from a service description example in [7].

Conjunction ($\wedge_\mu$) and negation ($\neg_\mu$) allow to compose inclusions and exclusions of desired temporal behavior Ψ. The terminals of the expression are propositions P, as well as $\top$ and $\bot$, which match all or no processes, resp. The existence of an activity of type A followed by the constraint Ψ, which must hold in the state subsequent to the activity, can be requested. The minimal fixpoint operator ($\mu X.\Psi(X)$) allows for the specification of formulas recursively. A proposition P is described with $\mathcal{SHOIN}(\mathbf{D})$ TBox axioms. Analogously, input and output parameter types and relationships, communication channels, effects of computational activities are described by $\mathcal{SHOIN}(\mathbf{D})$ TBox axioms.

Note, that disjunction, universal quantifier for actions as well as maximal fixpoint operator can be built using the above basic constructs. Furthermore, commonly used temporal constructs like Ψ_1 **until** Ψ_2, **eventually** Ψ, and **always** Ψ can be modeled with the help of the fixpoint operator.

The semantics $[\![\Psi]\!]_\mathcal{V}$ of a constraint Ψ is defined over an LTS $L = (S, T, \rightarrow)$ and a valuation $\mathcal{V}$ for interpreting propositions. In our case, $\mathcal{V}$ is the DL interpretation function, since our propositions are DL TBox axioms. A formula $\Psi$is interpreted as a set of states of the LTS. We refer to [3], [8] for further details on syntax and semantics of μ-calculus and that of its combination with $\mathcal{SHOIN}(\mathbf{D})$.

III. Annotation of Behavior Descriptions

Our first contribution is the extension of the behavior description formalism with the ability to annotate them with further constraints that cannot be expressed by formal process expressions using π-calculus process algebra combined with DL. We start motivating the need for behavior annotations and provide the semantics of annotated behavior descriptions afterwards.

Motivation: Behavior descriptions are suitable to describe complete processes. All observable details of the modeled system need to be known in order to describe an executable process expression based on the closed world assumption. This can be done by the provider, e.g. of an enterprise system, who can derive the detailed facts from the implementation. However, it is not realistic that any implementation details are known to Web users. E.g., it is not important and perhaps not possible to state which particular SSL certificate a Web site uses for the payment process. However, the information that the service supports SSL encrypted communication is relevant, but cannot be expressed at the ABox level of process models where the certificate instance is described. Rather, it needs to be expressed on a logic level by using existential quantifiers, e.g., declaratively expressed in the constraint formalism.

In contrast to behavior descriptions, the constraint language allows for the exclusion of behavioral properties. Side effects that are not observed by a user during execution can also be expressed by the constraint formalism. E.g., π-calculus based behavior description cannot express that a service will never pass user-provided payment information to any other party.

Because it is typically impossible to observe complete process behaviors in the Web and as there are behavioral

properties that cannot be captured by a process algebra, we extend the behavior description formalism by declarative constraints. Note, that modeling these properties is relevant in the context of search for behaviors of processes and services.

Annotation: We allow to annotate behavior descriptions with a set of behavior constraints using the constraint formalism introduced in Section II-B. An ordinary behavior description p_1 is interpreted by an LTS L_{p_1}. Let p_2 be a behavior description that is annotated with a behavior constraint Φ, we say $\Phi(p_2)$. Then p_2 is interpreted by $L_{p_2} \in \mathcal{L}_\Phi$, where $\mathcal{L}_\Phi$ denotes the set of LTS' that fulfill the constraint Φ. That is, the set of start states of the LTS' in $\mathcal{L}_\Phi$ equals the set of start states identified by $[\![\Phi]\!]$. Using the modelchecking technique [3] to check whether the constraints of a class are fulfilled by a behavior description, the consistency of class annotations is automatically assured if no contradictions exist. A behavior description p can also be annotated with several constraints $\Phi_1, ..., \Phi_n$ with $n \geq 1$ simultaneously. The semantics is defined as follows.

$$\Phi_1(p), ..., \Phi_n(p) \Leftrightarrow L_p \in \bigcap_{1 \leq i \leq n} \mathcal{L}_{\Phi_i}$$

IV. Behavior Specification with Behavior Classes

We now introduce the notion of behavior classes. We advance state of the art approaches with a hybrid description formalism for behavior classes and highlight the benefits while discussing their use in service modeling.

A. Behavior Classes

A behavior class is formally defined by a constraint Φ_c and a class name (label) c. The class represents a set of services or processes that share common behavioral attributes declared by the class definition Φ_c. The name is a human comprehensible textual representation of the asserted attributes and is solely used for the purpose of increasing the usability. Named classes have been widely used in taxonomies of products and services (e.g., UNSPSC) and are a fundamental concept in ontologies [9] in order to abstract from particular concept or role definitions. In contrast to a taxonomy without formal class definitions, our formal classification allows for automatic and consistent classification of behavior descriptions into existing classes. Also, tools are able to reason about the attributes asserted by a manual class assignment.

Behavior Class Syntax The specification of behavior classes is based on the constraint language introduced in Section II. For class definitions, we extend the given syntax of constraints by a choice for the inclusion of class names (C) as follows.

$$\Phi := C \mid \Phi \wedge_\mu \Phi \mid \neg_\mu \Phi \mid \mu X.\Phi(X) \mid \langle a \rangle \Phi \mid P \mid \top \mid \bot$$

Behavior Class Semantics The semantics $[\![c]\!]$ of a class c in a constraint is defined over the LTS $L = (S, T, \rightarrow)$ of a behavior description and corresponds to the set of states that fulfill the constraint Φ_c. That is, $[\![c]\!]_\mathcal{V} = [\![\Phi_c]\!]_\mathcal{V}$. We say that a service or process p is member of a behavior class c, iff the start state $s_0 \in S$ of the LTS L_p is in the set of states $[\![\Phi_c]\!]_\mathcal{V}$ that comply to the constraints of Φ_c.

Example A class named "*WebTrainBooking*" describes the generic train booking behavior and is defined by $\Phi_{c_{wtb}} = \phi \wedge_\mu$ **eventually** $\langle \mathsf{ch}\langle \mathsf{Ticket} \rangle \rangle \varphi$. The constraint $\Phi_{c_{wtb}}$ specifies assumptions ϕ about the parameters start, end, and departure of a trip given at the invocation of the process. ϕ may contains a proposition like $\mathsf{tc{:}trainTrip}(\mathsf{Trip}) \wedge \mathsf{tc{:}start}(\mathsf{Trip}, \mathsf{Start}) \wedge \mathsf{tc{:}end}(\mathsf{Trip}, \mathsf{End})$. Members of the behavior class *WebTrainBooking* must eventually return a Ticket via a communication channel ch. After the output took place, the following proposition φ must hold.

$$\begin{aligned} \varphi \quad = \quad & \big(\mathsf{po{:}dropShiftDelivery}(\mathsf{Delivery}) \vee \\ & \mathsf{po{:}onlineDelivery}(\mathsf{Delivery})\big) \wedge \\ & \mathsf{po{:}deliveryItem}(\mathsf{Delivery}, \mathsf{Product}) \end{aligned}$$

It states that the Product is either shipped via surface mail or email. The Product is further defined in DL to describe the relation between the produced Ticket and the given inputs.

B. Behavior Class Hierarchy

A class hierarchy $\mathcal{C} = (C, H)$ is a graph structure with a finite set of behavior classes C and the subclass relationship H over classes as the only relation. A class $c_i \in C$ is subclass of $c_j \in C$, we say $(c_i, c_j) \in H$, if all services and processes that are member of class c_i (i.e., model of the formal definition Φ_{c_i}) are also member of class c_j (i.e., model of Φ_{c_j}). H is a partial order that is not limited to be a tree structure; meshes are possible.

Given a set of behavior classes C, a hierarchy $\mathcal{C}$ is derived automatically based on class definitions Φ_c of each class $c \in C$. We compute the relation $H \subseteq C \times C$ by comparing pairs of class definitions Φ_{c_i}, Φ_{c_j} and determine the subclass relationship by adopting the technique from [10]. We first replace class names by their formal definition. Then, μ-calculus expressions are transformed into a normal form and the relationship between both expression is determined. A class c_i is a subclass of c_j iff any model of Φ_{c_i} is also model of Φ_{c_j}. In [11], the completeness of the axiomatization of the proposition μ-calculus presented in [10] has been shown.

Updates of the hierarchy are automatically managed due to the formal class definitions. The sub- and super class relationships between the new and existing classes are determined. Then, a new class is inserted as a subclass of the most specific super classes of the hierarchy. Changes in class definitions are treated analogously. When a class c is removed from a hierarchy, all subclasses of the deleted class c become subclasses of the super classes of c.

C. Simplification of Behavior Modeling

Modeling complex behavior as presented in Section II can become tedious. The complexity and length of behavior descriptions as well as the modeling effort are reduced by the use of classes without compromising on the expressivity. Users modeling the behavior can reuse existing classes along with further refinements for a specific behavior description instead

of a possibly very complex behavior specification formula. In collaborative service modeling, only domain or modeling experts will have the skills to model the service specifics beyond the scope of behavior classes. Since users are able to identify classes from a hierarchy by their descriptive name without the need to understand its formal definition, behavior descriptions can be created without any knowledge about the underlying formalisms. Discovery of appropriate classes currently remains a manual task including browsing of the hierarchy or keyword-based search that is an additional but less challenging effort for users. Then, the identified classes can be further inspected in order to verify that they match the user's intention. In our prototype (see Section VI-A), the keyword-based search for class names is made more flexible by searching also for class names containing hypernyms and hyponyms of the given keywords. Therefore, we use WordNet as dictionary. In the present work, we will not further focus on matching class names since it is not related to the contribution of this work.

Example We revive the above example and skip the extensive process modeling details and refer to an external resource[1] where we show the reduction of the modeling effort wrt. behavior descriptions. A train ticket behavior class such as *WebTrainBooking* is used to reduce the description complexity. While not all characteristics of a concrete service are covered by the generic class *WebTrainBooking*, it captures the common behavior of train ticket booking services and reduces the exemplified manual modeling effort for the service.

V. Exploiting Behavior Classes in Search

A. Behavior Classes in Requests

The request language as introduced in Section II constrains a desired behavior and, by this, spans a space of desired services or processes. An offered behavior matches a request if the offer equals a member of the set of desired behaviors. The reduction of the request modeling effort and simplification of requests is achieved by the reuse of behavior classes. The same benefits as for the simplification of behavior descriptions apply to requests, too.

In order to reuse existing behavior classes for expressing requests, appropriate behavior classes are derived by from an unstructured query from an end user, e.g., by mapping keywords to class names (as already discussed in Section IV-C). In this section, we consider formal requests, only.

The extended request formalism including behavior classes from Section IV is used for search queries, too. The formalism's semantics remains unchanged. Below, a request $\mathcal{R}$ for a desired train ticket booking behavior with an additional constraint on the acceptance of a credit card payment is shown. Parts of the desired behavior are inherited from the class definition Φ_{tb} of an existing *TrainBooking* behavior class shown below. $\mathcal{R}$ is described as an extension of the *TicketBooking* class by adding the constraint that the transacted purchase was provided by a railway company: $\Phi_{tb} = \textit{TicketBooking} \wedge_\mu \mathbf{eventually}\ \big(\mathsf{seller}(\mathsf{Purchase}, \mathsf{Seller}) \wedge \mathsf{RailOperator}(\mathsf{Seller})\big)$ The individual Purchase is already bound to this identifier in the definition of the class *TicketBooking*. The same identifier must be used in the definition of class *TrainBooking* and in further constraints in order to refer to the same individual. With the following example request, it becomes evident how much more complex the request $\mathcal{R}$ would be if no classes could be used for expressing the same set of constraints (that is, the definitions of *TrainBooking* and *TicketBooking* would be needed instead of the class name in the request below). Therefore, we argue that using classes tremendously simplifies and accelerates the specification of requests. As shown, the request can be further refined if the search result set is too coarse grained. Note, $\wedge$ denotes the logic conjunction with DL semantics while $\wedge_\mu$ was defined by the μ-calculus.

$$\begin{aligned}\mathcal{R} \;=\; & \mathbf{eventually}\ \big(\mathsf{po{:}seller}(\mathsf{Seller}) \wedge \\ & \mathsf{acceptsPaymentMethod}(\mathsf{Seller}, \mathsf{PayMeth}) \wedge \\ & \mathsf{pay{:}CreditCard}(\mathsf{PayMeth})\big) \wedge_\mu \textit{TrainBooking}\end{aligned}$$

B. Search Based on Off-line Classification

Now we exploit the behavior classes for an efficient search for processes and complex service behaviors. Besides the advantages of the hybrid behavior class formalism we have already shown, this section presents how the search performance is increased by formal behavior classes.

A search engine verifies constraints for each offered behavior individually and determines whether the offer fulfills them or not. For each behavior description with or without behavior class annotations, the search engine creates the corresponding LTS. Each state of an LTS is represented by a separate knowledge base. Due to the changes cause by state-changing actions during execution, the knowledge within different states can be contradicting and, thus, cannot be modeled in the same knowledge base.

We assume that the search engine has an LTS-representation of each behavior description in the repository $\mathbb{D}$. A classification hierarchy $\mathcal{C} = (C, H)$ with formal class definitions and the pre-computed hierarchic ordering over classes exists. During the initialization phase of the search engine, existing behavior descriptions are classified into existing classes automatically. Therefore, the expensive modelchecking technique determines whether behavior descriptions fulfill the constraints of a class or not and assigns them to behavior classes accordingly. Descriptions with class annotations are directly assigned to the resp. behavior classes.

The search engine materializes the classification information for its later use as an indexing structure $i \subseteq C \times \mathbb{D}$. For each behavior description $D_p \in \mathbb{D}$, we add (c, D_p) to the index i if the behavior p was determined to be a member of the behavior class $c \in C$. The task of building a hierarchy and classifying the behavior descriptions into the behavior classes is done off-line and is not considered as a part of the query answering task. The classification index i allows to request behavior descriptions of desired classes immediately. By this, it saves expensive logic modelchecking operations at query answering time.

Let a request $\mathcal{R}$ specify constraints by means of desired behavior classes $C_{\mathcal{R}} \subseteq C$ of the hierarchy. In addition, $\mathcal{R}$ contains further requirements which are explicitly modeled and not captured by the classes $C_{\mathcal{R}}$ (as in our example above). In a first step, the search engine retrieves behavior descriptions $i(C_{\mathcal{R}})$ from the index i. The first step returns all behavior descriptions that are member of all desired classes. If a request contains no behavior classes, the first step returns all descriptions from $\mathbb{D}$. This result serves as input for the second step, where further explicit requirements from $\mathcal{R}$ (that were not captured by the classes) are verified. The modelchecker iterates over each LTS L_p of the behavior descriptions passed from the first step and determines the set of states of the LTS L_p in which the constraints of the request $\mathcal{R}$ are satisfied. It breaks down composite into atomic formulas and aggregates the individual results recursively. E.g., for an atomic constraint P as part of $\mathcal{R}$, the matchmaker iterates over the states of the LTS and adds these states to the result set in which the proposition P holds. A detailed explanation of the modelchecking algorithm was given in [3]. When the verification of a complete request $\mathcal{R}$ is terminated, the modelchecker determines whether the offered service behavior satisfies the requested behavior which holds iff the start state of the LTS L_p is in the set of states returned by the modelchecker for $\mathcal{R}$.

By the introduction of the index i and the use of behavior classes, the number of behavior descriptions considered in the second step for expensive modelchecking operations is reduced. Further, the amount of constraints that are evaluated at query time is reduced. Therefore, we assume that retrieving instances that are member of a class or of several classes simultaneously is faster than verifying all constraints at query time. Obviously, many factors like the complexity of class definitions, the size of the classification, and the number of behavior descriptions have to be considered to let this assumption hold.

VI. Implementation and Evaluation

In this section we describe the implementation of the presented approach. The implementation is an integral part of the suprime framework[2] for intelligent usage and management of services and processes. Afterwards, we describe the setup of the experiments conducted in order to evaluate our claims regarding the performance gain and discuss the impact of the class granularity based on the measured results.

A. Implementation

1) Modeling Processes, Behavior Classes, and Queries: We developed Java APIs for modeling and annotating executable behaviors, and for specifying search queries for services and processes with matching behavior. For the DL part, we use the OWL API[3] for semantically describing the processes resources, and HermiT OWL reasoner[4] for reasoning about such semantically described resources.

Semantic behavior descriptions can also be created graphically using our process modeling tool (part of the suprime framework). For modeling behavior classes graphically, we provide an assisted form based input mask that allows the user to easily enter constraints like the existence of an input activity. Existing classes are displayed in a tree-shaped structure and can be selected for reuse.

2) Search: Search queries are passed to the search engine. In a first step, the search space is reduced by retrieving behavior descriptions from requested classes. This is done by invoking a HermiT Reasoner instance with the repository ontology $O_{\mathbb{D}}$. For a given hierarchy and a set of behavior descriptions, the ontology $O_{\mathbb{D}}$ models (1) each behavior class as an OWL class, (2) the hierarchical relationships of the behavior classes as OWL subclass-relations, and (3) each behavior description as an individual. The individuals are member of those OWL classes that correspond to behavior classes for which the behavior description is a model of. When the search engine is initialized, the repository ontology $O_{\mathbb{D}}$ is created and loaded. A request is processed as described in the previous section. First, the instances of desired classes are retrieved from $O_{\mathbb{D}}$. Then, further constraints that were not covered by the requested behavior classes are verified on the behavior descriptions retrieved in the previous step. The results of the second step, i.e. services or processes that fulfill all the constraints of the query are displayed to the user in the implementation of our graphical search interface.

B. Evaluation

We conducted several experiments that show the benefits of the presented approach wrt. search performance. In our experiments we examine the impact of the classification hierarchy on the search performance. As we apply logic-based model-checking and classification techniques, it is not necessary to evaluate the quality (soundness and completeness) of search results. Instead, we compare the query answering times for equivalent queries with and without the use of behavior classes. Further, we differentiate between various class complexities and class hierarchy sizes. We developed a class hierarchy with formal class definitions and measured the search performance for varying behavior class granularity. This means, given a fixed number of services or processes, applying a hierarchy with coarse-grained classes corresponds in average to a small hierarchy (number of classes), and few formal constraints per class. In contrast, fine-grained classes lead to a larger hierarchy, more constraints per class.

1) Test Data: The test data is derived from given descriptions of processes that coordinate existing Web based services. We use CoScripts from the IBM CoScripter repository[5]. CoScripts describe executable processes. The script language is directly mapped to our behavior description language. Our analysis of CoScripts from different domains like travel and

[2]https://km.aifb.kit.edu/projects/suprime/

[3]OWL API is available at http://owlapi.sourceforge.net.

[4]Hermit OWL reasoner is available at http://hermit-reasoner.com.

[5]The IBM CoScripter repository at http://coscripter.researchlabs.ibm.com/coscripter provides approx. 5800 scripts for automating Web processes.

TABLE I: Behavior description characteristics per process.

Process Behavior Characteristics	Range
Number of activities	3 ± 1
Number of parameters per I/O activity	3 ± 2
DL Axioms per I/O parameter	3 ± 2
Behavior class annotations	3 ± 2

TABLE II: Different levels of behavior class granularities.

Class Granularity	Fine	Medium	Coarse
Number of behavior classes	60	40	20
Simple constraints per class	3	2	1

real estate[6] resulted in the observation that many end user browsing processes are rather short comprising a small number of activities performed sequentially. The reason is that most of the currently available CoScripts automate interactions with only one Web site. More precisely, scripted travel processes, e.g., for flight or hotel offers typically involve approximately 3 (mainly input) activities (with about 3 parameters per input operation in average). The desired information within the returned Web page was often not explicitly extracted for further processing. So, there is only one parameter per output activity on average.

Based on the analysis of the CoScripts, we generated semantic behavior description within the characteristics summarized in Table I. The correctness wrt. to content of existing CoScripts cannot be guaranteed, as we do not focus on automatic learning of semantic behavior descriptions. Still, we can argue that the behavior complexity of our test data corresponds to the complexity of the behaviors described by the CoScripts. In our experiments we used 2000 generated descriptions. In average, our behavior descriptions describe each input/output parameter by three DL Axioms (i.e., at least its data type plus its relationship to other process resources). In Table II, the numbers of classes are related to the number of simple constraints. A *simple constraint* can be one of $\mathbf{eventually}\ \phi$, $\mathbf{always}\ \phi$, where ϕ is a simple constraint like a proposition P or the existence of an activity a ($\langle a \rangle P$).

We created several search queries composed ($\wedge_\mu$, $\vee_\mu$, $\neg_\mu$) out of 9 simple constraints. Analogously to the descriptions, the complexity of each proposition and parameters of desired activities is set to an average of 3 ± 2 DL axioms (class and object property assertions, e.g., $P \equiv \mathsf{Flight(f)} \wedge \mathsf{Time(t)} \wedge \mathsf{departureTime(f,t)}$)[7]. A desired action is expressed by its type (class assertion) and the description of types and relationships between messages.

2) Results: Tables IIIa and IIIb show the measured query answering time for varying class granularities and proportion of behavior classes used in the queries for 1000 and 2000 behavior descriptions respectively. The baseline are queries Q_3 in which no behavior classes are used. The opposite extreme is Q_0 which consists of a combination of classes only. The relative search performance gain is expectedly high.

TABLE III: Query answering time and relative gain.

(a) 1,000 behavior descriptions

Class Granularity	Fine		Medium		Coarse	
	time [s]	gain	time [s]	gain	time [s]	gain
Q_0 (classes only)	0.010	99%	0.009	99%	0.009	99%
Q_1 (66% classes)	3.32	61%	3.21	64%	3.35	63%
Q_2 (33% classes)	5.46	37%	6.10	32%	6.28	30%
Q_3 (no classes)	8.63	0	9.02	0	9.05	0

(b) 2,000 behavior descriptions

Class Granularity	Fine		Medium		Coarse	
	time [s]	gain	time [s]	gain	time [s]	gain
Q_0 (classes only)	0.012	99%	0.018	99%	0.019	99%
Q_1 (66% classes)	10.47	67%	11.83	65%	11.21	68%
Q_2 (33% classes)	19.22	39%	22.06	34%	24.01	31%
Q_3 (no classes)	31.32	0	33.40	0	34.98	0

An interesting outcome of this experiment is that the class granularity is not crucial to the absolute query answering times and the relative gains. Although a slight but steady performance decrease is measured for coarse grained classes, it shows that the overhead of fine grained classes (i.e., having many classes in a hierarchy) does not add significant penalties wrt. search performance. This observation is attributed to efficient instance retrieval provided by ontology reasoners in case of a relatively large ABox as compared to the TBox. Consequently, the experiments show that a hierarchy can easily grow as can be expected for a Web of services. This argument is underpinned by applicable optimizations that exist for this particular reasoning task [12]. The off-line computation of behavior class memberships further reduces the instance retrieval time. Furthermore, the query answering time increases significantly for larger numbers of available behavior descriptions. The query answering takes up to 3.9 times longer for 2000 descriptions than for 1000 descriptions if no classes are used (Q_3). If only classes are used (Q_0), the factor can be reduced to 1.2. Thus, the importance of the performance gain by classes becomes even more important when the number of behavior descriptions increases.

VII. Related Work

A. (Semantic) Business Processes

In [13], a process query language (PQL) has been introduced. PQL is based on an interpretation of process models as entity-relationship diagrams. A query is a regular expression that allows the usage of '*' for the occurrence of sub-task relationships. Apart from the lack of temporal operators needed for reasoning about the process behavior, the pattern matching based query answering algorithm cannot find answers (process models) that use syntactically different terminology than the one used in the query.

BPMN-Q [14] is a graphical query language that uses concepts and notations from BPMN. BPMN-Q matches a query graph pattern against a process graph and considers control flow and names of the activities. This approach can be

[6]We used the repository's keyword based search facilities which by no means guarantees the completeness of the search results.

[7]The travel domain ontology is available at http://www.w3.org/2000/10/swap/pim/travelTerms.rdf.

compared to our model checking technique developed in our previous works except that with our model checking technique we are able to reason about the data flow and resources as well.

The Business Process Execution Language for Web Services (BPEL4WS) [15] can be compared to our language for describing executable observable behavior. In this paper, our focus was on the usage and impact of behavior classes on process descriptions and search. Therefore, our constraint specification formalism (with or without classes) can be used for compliance checking and search for BPEL4WS process models if the latter could be interpreted as an LTS.

There are a couple of efforts for adding semantics to the business processes by aligning process resources to domain ontologies and reasoning over the process with an ontology reasoner. However, these approaches have to rely on treating transitions between states as logical implications within a state, which restricts them to only those processes that have a priori known fixed number of states. A process algebra for modeling complex process behavior using ontologies for semantically describing the process resources was proposed in [6]. However, models described with process algebras need to be treated with closed world semantics, which is often a restriction in case of Web-based processes. Therefore, we advanced this approach by capability to add declarative constraints to process models.

B. Semantic Web Services

The *OWL-S Service Profile* aims at the description of atomic services and also implements the concept of a service classification [16]. However, the relation between service classification and the rest of the service description is not exploited in existing OWL-S based matchmakers (e.g. [17]). The *Process Model* supports specification of complex behavior. In contrast to the Service Profile, the Process Model does not directly support the use of classification. Even though a service classification can be introduced by adding subclasses to this class Service, it is not possible to reason about the dynamics of complex service behavior due to missing formal execution semantics of the OWL-S Process Model.

WSMO [18] supports the semantic description of both atomic and complex services. WSMO-Lite [19] is a vocabulary for annotating (SA)WSDL [20] service descriptions with semantically described service properties, e.g. a subclass of FunctionalClassificationRoot. However, as in OWL-S Profile, functional classification is isolated from other functional properties such as preconditions and effects. Thus, service classifications contradicting the remaining service description are possible. An approach presented in [21] uses functional classifications to achieve efficient discovery of atomic Web services. Furthermore, the formal class definitions are not allowed to be used in combination with formal query parts in a requests.

C. Further Service Modeling Approaches

UDDI [22] allows for coarse-grained service discovery based on the classification meta data. It still suffers from underspecified classes from standardized or proprietary classifications and also from an XML based data model which lacks explicit semantics. Such classification taxonomies like the United Nations Standard Products and Services Code (UNSPSC) were also used in OWL-S service matchmakers [17]. Then, a matchmaker simply evaluates class assignments to determine matches but no class semantics is given or used.

Corella et al. also indicate the potential of service classifications that offer an intuitive and coarse-grained service retrieval mechanism by investigating how services can be semi-automatically classified based on the similarities to previously classified services [23]. A Bayes-based method to classify services automatically [24] also promises high accuracy but lacks support for logical consistency since the classes were only described by class identifiers and not formally defined.

Based on the assumption that matchmaking of desired service classification with offered service classifications is less expensive than the DL-based matchmaking of the functionality description, Lara et al. presented a service discovery approach in [25], with the help of formally defined service classes with WSML-DL+. In our approach, we can use classes in service descriptions when concrete property values are not known or cannot be revealed. Furthermore, we allow usage of classes in a request together with further constraints, whereas the approach presented in [25] supports only predefined classes in a request.

Our modeling approach is similar to the state-transition system based process modeling approach for BPEL4WS presented in [26], as well as to the Finite State Machines (FSM) based approach for modeling Web services presented in [27]. [28] presents a Linear Temporal Logic based approach for verifying business constraints at runtime. In contrast to our approach, in which we describe and reason about the data content of a state semantically with description logics, none of them allows reasoning about the data objects involved in a process. Still, our behavior classes can be used to classify and find BPEL4WS processes and Web services as well, given that a semantics as a mapping to an LTS is provided.

A pragmatic temporal logic for reasoning about time intervals of events by proposing a set of relationships between time intervals was introduced in [29]. Even though there are no results about the expressivity of the interval relationships to the best of our knowledge as well as the fact that μ-calculus is point-based, we believe that the interval relationships can be expressed by μ-calculus. While the interval relationships due to their pragmatic meaning could serve as a good basis for obtaining the end user query language, there are differences in the semantics that need to be taken care of while defining the translation of an interval relationship to a μ-calculus formula. An example for such a difference is "negation". In interval logic negation of "before" is "after", while in μ-calculus negation of "before" is "not before", which has a different meaning than "after".

Our query specification formalism falls in the category of DL with modal operators [30]. Since we use constant domain terminologies (expressed as DL TBox), we are able to use a

more expressive temporal logic than the ones discussed in [30] while still ensuring decidability.

VIII. Conclusions and Future Work

Our work was primarily motivated by our observation that despite most of the interesting services are not atomic but rather have complex behavior, there are hardly any convenient specification and search techniques available for such services. In this paper, we showed how the description of observable service behavior can be annotated with behavior specification formulas. Then, we introduced the notion of behavior classes that have human comprehensible names as well as formal definitions. We showed how behavior classes can be exploited for annotating the behavior descriptions as well as in process search queries. Furthermore, we exploit the behavior class hierarchy for pre-computing and cache class memberships of descriptions in order to save on-line query time. We have implemented the presented specification and search approach and showed with our evaluation results the positive impact of behavior classes on the search performance.

In the future we will focus on the exploitation of the formal Web process descriptions that leads to the automatic orchestration of them for the creation of more complex Web applications. Web processes with similar functionalities can be automatically aggregated in order to allow users to gain a broad overview. Due to the automation, Web applications like travel booking or price comparison portals can be created dynamically for any domain. A convenient query language and the automatic derivation of structured queries will be investigated to address practicability. Finally, an efficient retrieval technique is the prerequisite for an efficient composition. Therefore, we wish to investigate (1) the automatic creation of behavior classes by using frequent request patterns and clustering existing service descriptions, (2) increased use of classes in requests by query rewriting, and (3) hierarchy update strategies for dynamically updating the classification hierarchy.

Acknowledgment

This work was supported by the German Federal Ministry of Education and Research (WisNetGrid, Grant 01IG09008).

References

[1] R. F. Bruce and J. M. Wiebe, "Recognizing Subjectivity: A Case Study of Manual Tagging," *Natural Language Engineering*, vol. 5, no. 2, pp. 187–205, 1999.

[2] S. Agarwal and M. Junghans, "Meaningful Service Classifications for Flexible Service Descriptions," in *SERVICES*. IEEE Computer Society, 2011, pp. 85–86.

[3] S. Agarwal, S. Lamparter, and R. Studer, "Making Web services tradable - A policy-based approach for specifying preferences on Web service properties," *Web Semantics: Science, Services and Agents on the World Wide Web, Special Issue on Policies*, vol. 7, no. 1, pp. 11–20, 2009.

[4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[5] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes, Parts I and II," *Journal of Information and Computation*, vol. 100, pp. 1–77, 1992.

[6] S. Agarwal, S. Rudolph, and A. Abecker, "Semantic Description of Distributed Business Processes," in *AAAI Spring Symposium: AI Meets Business Rules and Process Management*. AAAI, 2008, pp. 1–11.

[7] D. Fensel, F. M. Facca, E. P. B. Simperl, and I. Toma, *Semantic Web Services*, 1st ed. Springer, 2011.

[8] C. Stirling, *Modal and Temporal Properties of Processes*. Springer, 2001.

[9] S. Staab and R. Studer, Eds., *Handbook on Ontologies*, 2nd ed., ser. International Handbooks on Information Systems. Springer, 2009.

[10] D. Kozen, "Results on the Propositional mu-Calculus," *Theoretical Computer Science*, vol. 27, pp. 333–354, 1983.

[11] I. Walukiewicz, "A note on the completeness of Kozen's axiomatisation of propositional mu-calculus," *Bulletin of Symbolic Logic*, vol. 2, no. 3, pp. 349–366, 1996.

[12] V. Haarslev and R. Möller, "On the Scalability of Description Logic Instance Retrieval," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 99–142, 2008.

[13] M. Klein and A. Bernstein, "Toward High-Precision Service Retrieval," *IEEE Internet Computing*, vol. 8, no. 1, pp. 30–36, 2004.

[14] A. Awad, G. Decker, and M. Weske, "Efficient Compliance Checking Using BPMN-Q and Temporal Logic," in *BPM*, ser. LNCS, M. Dumas, M. Reichert, and M.-C. Shan, Eds., vol. 5240. Springer, 2008, pp. 326–341.

[15] *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, OASIS, 2007.

[16] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, "Automated Discovery, Interaction and Composition of Semantic Web Services," *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 1, pp. 27–46, 2003.

[17] N. Srinivasan, M. Paolucci, and K. Sycara, "Semantic Web Service Discovery in the OWL-S IDE," in *HICSS*. IEEE Computer Society, 2006.

[18] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, "Web Service Modeling Ontology," *Applied Ontology*, vol. 1, no. 1, pp. 77–106, 2005.

[19] T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel, "WSMO-Lite Annotations for Web Services," in *ESWC*, ser. LNCS, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, Eds., vol. 5021. Springer, 2008, pp. 674–689.

[20] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell, "SAWSDL: Semantic Annotations for WSDL and XML Schema," *IEEE Internet Computing*, vol. 11, no. 6, pp. 60–67, 2007.

[21] M. Stollberg, M. Hepp, and J. Hoffmann, "A Caching Mechanism for Semantic Web Service Discovery," in *ISWC/ASWC*, ser. LNCS, K. Aberer, K.-S. Choi, N. F. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds., vol. 4825. Springer, 2007, pp. 480–493.

[22] "UDDI Spec Technical Committee Draft 3.0.2," http://uddi.org/pubs/uddi_v3.htm, OASIS, OASIS Committee Draft, 2004.

[23] M. Á. Corella and P. Castells, "Semi-automatic Semantic-Based Web Service Classification," in *Business Process Management Workshops*, ser. LNCS, J. Eder and S. Dustdar, Eds., vol. 4103. Springer, 2006, pp. 459–470.

[24] G. Li, W. Zhang, H. Li, and J. Guo, "An Efficient Way to Accelerate Service Discovery and Invocation," in *SMC*. IEEE, 2007, pp. 1304–1309.

[25] R. Lara, M. Corella, and P. Castells, "A Flexible Model for Web Service Discovery," in *1st International Workshop on Semantic Matchmaking and Resource Retrieval: Issues and Perspectives*, Seoul, Korea, 2006.

[26] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi, "Automated Synthesis of Composite BPEL4WS Web Services," in *International Conference on Web Services (ICWS)*. IEEE Computer Society, 2005, pp. 293–301.

[27] R. R. Hassen, F. Toumani, and L. Nourine, "Web services composition is decidable," in *WebDB*, Vancouver, BC, Canada, June 2008.

[28] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, "Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata," in *BPM*, ser. LNCS, S. Rinderle-Ma, F. Toumani, and K. Wolf, Eds., vol. 6896. Springer, 2011, pp. 132–147.

[29] J. F. Allen, "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[30] A. Artale and E. Franconi, "A survey of temporal extensions of description logics," *Annals of Mathematics and Artificial Intelligence*, vol. 30, no. 1-4, pp. 171–210, 2001.