Efficient Search for Web Browsing Recipes

Martin Junghans
Institutes AIFB and KSRI
Karlsruhe Institute of Technology
Englerstr. 11, 76131 Karlsruhe, Germany
Email: junghans@kit.edu

Sudhir Agarwal
Stanford Computer Science Department
Stanford University
353 Serra Mall, Stanford, CA 94301, USA
Email: sudhir@cs.stanford.edu

***Abstract*—Information search in the Web can become cumbersome if the desired information is scattered across multiple websites. For instance, even though there exist pages listing track chairs of the past ICWS conferences and web accessible bibliography databases, compiling the list of recent journal publications of the ICWS track chairs with the help of existing search engines is still a time consuming task. It is even harder to find information from the Deep Web as it requires user interactions that are hard to simulate by automatic crawlers.**

Semantic search based on structured data aims at efficiently answering information needs but relies on the cooperation of providers to be able to access their data. We aim at providing an alternative solution by introducing web browsing recipes that are goal-oriented end user browsing processes containing instructions for accessing, extracting, and merging (dynamic) information from various websites. Browsing recipes can be shared and reused to allow users to benefit from the browsing efforts of others. In order to achieve this goal, the development of efficient search techniques is the main prerequisite for effectively sharing and reusing recipes.

In this paper we propose an efficient search technique for finding browsing recipes from large recipe repositories. Our search technique necessitates a structured query specifying the required information along with constraints on the structure of the browsing processes. We augment explicit state representation based model checking technique by indexing structures tailored to the requirements of information search based on the recipes. The performance evaluation of our approach reveals the impact of the indexing structures on the overall recipe search efficiency.

***Keywords*-model checking; indexes; services; web search**

I. Introduction

For many practical purposes end users need information that is scattered across multiple websites. The websites can be static or dynamic. Static websites can be crawled by current search engines, and their content can be indexed to provide end users with efficient search over documents. However, in many cases, end users still have to do a lot of work manually to compile the required information together. E.g., consider an end user who is interested in the track chair names of previous ICWS conferences. As of today, Google does not deliver satisfactory results for queries similar to "track chairs of all ICWS conferences". One reason for this lack of support is that the set of links returned by document-centric web search engines often contain similar information whereas the complex information need requires fractions of complementary information that, if combined, satisfy the information need. In order to obtain the required information the end user has to pose multiple queries to a search engine, browse through the results, extract and aggregate the required information fragments outside of the found web pages.

The case of dynamic websites is even more complex. Accessing the information lying in the Deep Web [1] is already an open challenge for search engines. It is not trivial for automatic crawlers to sensibly interact with the dynamic websites in order to access the underlying information. Furthermore, indexing such information is not a suitable technique since the information underlying dynamic websites can change so rapidly that the index becomes quickly outdated.

End users need help in selecting the pages that are relevant for obtaining the information scattered over multiple web pages. Such a help must contain at least the set of the pages that the end user should visit, and support for invoking the pages in the set easily. More advanced help could comprise the complete end user browsing process including support for data flow between the user and the pages as well as among the pages, and control flow if there are data dependencies among inputs and outputs of web pages in the set.

The Semantic Web [2] as well as the Linked Data [3] addresses the issue of semantic descriptions of web page content but not of the path to be executed in order to reach pages. Information retrieval has focused on analyzing such browsing paths or click trails of millions of users mainly for the purpose of improving web search results. Click trails can be used as endorsements to rank search results more effectively [4], trail destination pages can themselves be used as search results [5], and the concept of teleportation can be used to navigate directly to the desired page [6]. The statistics based click analysis methods typically fail to consider semantics of user queries and pages. Furthermore, the models cannot differentiate whether a frequently used path actually satisfies the information need or not.

Our Approach: We aim at providing end users with a list of browsing processes that are relevant for a given information need instead of a list of links to web pages. Each browsing process in the list of hits will lead the user to the required information. Such a list of appropriate browsing processes is computed by searching existing end user browsing processes.

In order to be able to search appropriate browsing processes automatically, we first show in Section II how end user browsing processes (consisting of link selection, form

inputs, and information extraction steps) can be formalized by combining a process algebra with a semantic description of process resources [7]. We also show how browsing processes can be automatically verified with a model checking technique against search requests. Requests declaratively describe the required information as well as further constraints on the process structure using a combination of a description logic and a temporal logic. In our approach we can leverage web page annotations, but we do not require pages to be previously annotated. Explicit end user browsing processes provide the end users with a place for adding the semantic annotations to the web pages they contain in a bottom up fashion. Then, we show how user browsing processes can be efficiently searched so that users can save browsing time by reusing them for their complex information gathering needs. Existing model checking implementations often implement an explicit state representation and fall short of efficiency, which is required for the search over large repositories. Therefore, we develop two offline indexes in Section III that achieve significant gains in the search performance. In this section, we will present evaluation results of performance tests and complexity proofs to demonstrate the impact on search performance. In Section IV, we provide implementation details, experimental setup and test data. After discussing related work in Section V, we conclude in Section VI.

II. Preliminaries

We start with brief overviews of the basic techniques that we use throughout the paper. We present how browsing processes can be described formally, how constraints on such processes can be specified, and how processes can be automatically checked against constraints with a model checking approach. In contrast to the imperative process descriptions, declarative constraints over processes as we use them in search requests express desired behavioral properties. Model checking techniques verify whether a given behavior description satisfies a given set of constraints and serve as a fundamental reasoning task for searching for complex behavior descriptions of browsing recipes.

To illustrate the formalisms, we will use the example of browsing processes that collect the track chairs of previous conferences and query their articles from a bibliography database. In the example, static conference web pages are visited and the names of track chairs are marked as relevant outputs. Next, a website like DBLP is visited, author names are entered into the input form of the start page and the corresponding articles are extracted from the result pages.

A. Semantic Process Description Language

We use the suprime Process Description Language (*suprimePDL*) to describe the information flow and control flow among the web pages. *suprimePDL* is based upon the π-calculus process algebra [8]. Process algebras allow for the description of observable behavior by providing constructs for modeling data and control flow. Here, concrete actions with concrete parameter values occurring in process runs are aggregated to variables. The process language contains input and output activities, local operations (computation), conditionals, parallel compositions, alternatives, and agent invocations. We explain the syntax in the following examples as needed.

In pure π-calculus [8], the process resources and variables are seen as strings without any structure. As a result, it is hard for users to understand which values he should provide for the variables in order to get the desired result. In *suprimePDL*, process resources and variables are annotated with a domain ontology O_D expressed in RDFS. E.g., input parameters $\mathbf{x}$ and the communication channel c of an input activity $\mathsf{c}[\mathbf{x}]$ are process resources and further described in O_D. By combining a process description language with a description logic (DL) [9], e.g. $\mathcal{ALC}$, we cannot only describe the types of process resources and variables but also how they relate with each other [7]. E.g., for an input activity with two parameters of type 'Peson' and 'Publication', we can describe that one parameter denotes the author of the publication.

The semantics of *suprimePDL* process expressions is defined on a labeled transition system (LTS) [7]. The states of an LTS correspond to the knowledge of the process in that stage of the execution, and the transitions correspond to the atomic (input, output, or local) activities. The ABox of the state knowledge is subject to change during state transitions and the TBox is assumed to be invariant during execution.

Definition 1 (labeled transition system): For a set of atomic propositions P and a set of actions A, an LTS is a tuple (S, T, A, λ), where S is a finite set of states, $T \subseteq S \times A \times S$ a set of labeled transitions between the states, and $\lambda : S \to 2^{AP}$ a labeling function that maps each set $s \in S$ to the set of atomic propositions that are true in s.

1) Modeling Browsing Processes with suprimePDL: A single web page is a message sent by its hosting server to an end user. In addition to the information content, a web page may contains the description of a choice process. The choice process, denoted by $+$, consists of a set of links and a set of forms. Formally, the output activity $y\langle\mathbf{v}\rangle$ of the server that produces a web page with l values $v_1, \ldots, v_l$, m links $l_1, \ldots, l_m$ and n forms $f_1, \ldots, f_n$ can be described as:

$$y\langle v_1, \ldots, v_l\rangle.@L_1\{\mathbf{x_1}\} \quad + \quad \ldots + @L_m\{\mathbf{x_m}\} \\ + \quad @F_1\{\mathbf{y_1}\} + \ldots + @F_n\{\mathbf{y_n}\},$$

where $L_1, \ldots, L_m$ denote the base URLs of the links $l_1, \ldots, l_m$, $\mathbf{x_1}, \ldots, \mathbf{x_m}$ their parameters if any, $F_1, \ldots, F_n$ the action URLs of the forms $f_1, \ldots, f_n$, and $\mathbf{y_1}, \ldots, \mathbf{y_n}$ their submission parameters. In our view, a URL is equivalent to an agent identifier, whereas the selection of a link is equivalent to an agent invocation (denoted by a @ preceding an identifier) with concrete values for the arguments.

We model the link arguments as classes in the ontology associated with the process expression describing its base URL, and the values of the arguments as instances of the classes corresponding to the arguments. A form corresponds to a complex ontology class. Names of the form's input elements

correspond to the properties of the complex class. The name of the class corresponding to the range of a property can be often derived from the label of the input field (see e.g. [10]). Some types of input elements provide a set of values from which one or more values can be selected. In these cases, the provided values are modeled as ontology instances, while the class representing the range of an input field as an RDFS container class instead of a normal class.

A browsing process is equivalent to an agent identifier that is defined as a parallel composition of the invocations of the agent identifiers $P_1, \ldots, P_n$ corresponding to the websites visited in the process and a coordinating process C. Such a process is defined as $@C\{\} \parallel @P_1\{\} \parallel \ldots \parallel @P_n\{\}$.

Example 1: A conference website at URL u_1 links among many other options to the call for research papers web page denoted by the process CFP_{icws13}. The process provides a web page listing all research tracks **tracks** denoted by the output activity $u_1\langle\mathbf{tracks}\rangle$.

$$\mathit{CFP}_{\text{icws13}}() \equiv u_1\langle\mathbf{tracks}\rangle.\sum_{t\in\mathbf{tracks}} @\mathit{CFP}_t\{\}$$

Selecting one of the provided links, say $@CFP_{\text{research}}\{\}$, returns the page about one specific track and lists topics, track chairs, and PC members in another output activity.

$$\mathit{CFP}_{\text{research}}() \equiv u_1\langle\mathbf{topics}, \mathbf{chairs}, \mathbf{pcs}\rangle.\mathbf{0}$$

Figure 1 shows the corresponding LTS representation of the browsing process for accessing the track information of the conference. Different heterogeneous processes with similar outcomes may exist.

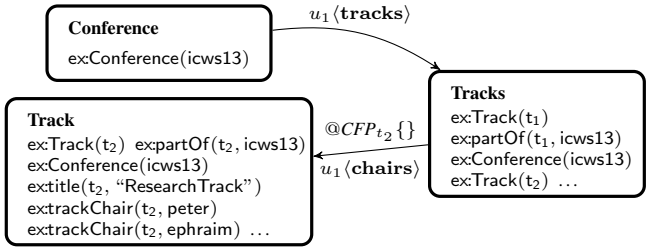

Fig. 1. Excerpt of the LTS of a recipe collecting track information.

The second website to be considered in this example is DBLP at u_2 in order to search for publications. Among other information on the entry page $DBLP()$, the web form $Search$ takes the author name obtained from the input activity as single value.

$$\mathit{DBLP}() \equiv u_2\langle\mathbf{x}\rangle.u_2[author].@Search\{author\}$$

The form submission generates a dynamic page listing the publications of the specified $author$. Although this page contains many further links, we restrict our example and terminate the process at this stage with the Null process.

$$Search(author) \equiv u_2\langle\mathbf{articles}, \mathbf{authors}, \ldots\rangle.\mathbf{0}$$

The whole browsing process $Recipe$ then combines both information sources by invoking both components and coordinating the flow of the track chair names from $\mathit{CFP}_{\text{research}}$ to DBLP. This is achieved by means of a controlling process C that runs parallel to both component websites, receives the outputs of the conference website, provides the author names to DBLP, and receives the list of publications **pubs** of each track chair.

$$Recipe() \equiv @C\{\} \parallel @\mathit{CFP}_{\text{icws13}}\{\} \parallel \prod_{c\in\mathbf{chairs}} @\mathit{DBLP}\{\}$$

$$C \equiv u_1[\mathbf{chairs}].\prod_{c\in\mathbf{chairs}} u_2\langle c\rangle.u_2\langle\mathbf{pubs}\rangle.\mathbf{0}$$

2) Capturing Browsing Processes: A major advantage of our approach is that it ensures that the capturing of browsing processes does not require extra manual effort from the end users. Existing browser plugins like the open source CoScripter[1] and the commercial iMacros[2] can record users browsing actions and save them as executable scripts. Rather than the top-down approach of ontologies in the Semantic Web, we advocate a bottom-up approach beginning with websites and the end users who browse them using standard browsers. Website owners are not involved, and annotations consist of exactly those necessary for a particular application. Once a user has tediously found the right path through a sequence of websites for a particular goal, he should be able to find it much faster and easier the next time. Saving and sharing the process in a reusable way is thus valuable.

B. Model Checking Browsing Processes

Model checking has been mainly used to verify hardware and software systems. A model checker takes a formula in a constraint specification language (μ-calculus with description logic in our case) and a system or its model (described with π-calculus and description logic), and checks automatically whether the system or the model satisfies the formula.

1) Specification of constraints: Constraining the temporal behavior is done on the basis of a temporal logic. The syntax of the behavior constraint language that we will use is:

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

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, variables Z, as well as $\top$ and $\bot$. $\top$ and $\bot$ match all or no processes respectively. 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 $\mathcal{ALC}$ (attributive concept language with complements [11]) axioms. Analogously, input and output parameter types and relationships, communication channels, effects of computational activities are described by

[1] http://coscripter.researchlabs.ibm.com/coscripter
[2] http://www.iopus.com/iMacros

Algorithm 1: evaluateFormula

Require: Formula Ψ and LTS $(S, T, A, \mathcal{V})$
1: **if** $\Psi = \top$ **then**
2: **return** S
3: **else if** $\Psi = \bot$ **then**
4: **return** $\emptyset$
5: **else if** $\Psi = P$ **then**
6: $S' \leftarrow \emptyset$
7: **for all** $s \in S$ **do**
8: **if** *evaluateProposition*$(P, s) = true$ **then**
9: add s to S'
10: **return** S'
11: **else if** $\Psi = \Psi_1 \wedge_\mu \Psi_2$ **then**
12: **return** *evaluateFormula*$(\Psi_1) \cap evaluateFormula(\Psi_2)$
13: **else if** $\Psi = \neg_\mu \Psi_1$ **then**
14: **return** $S \setminus$ *evaluateFormula*(Ψ_1)
15: **else if** $\Psi = \langle a \rangle \Psi_1$ **then**
16: $S' \leftarrow \emptyset$
17: **for all** $(s_1, a', s_2) \in T$ **do**
18: **if** $a' = a$ and $s_2 \in$ *evaluateFormula*(Ψ_1) **then**
19: add s_1 to S'
20: **return** S'
21: **else if** $\Psi = \mu Z.\Psi_1(Z)$ **then**
22: $Z \leftarrow \emptyset$
23: **repeat**
24: $Z' \leftarrow Z$
25: $Z \leftarrow$ *evaluateFormula*$(\Psi_1(Z))$
26: **until** $Z' = Z$
27: **return** Z
28: **else if** $\Psi = Z$ **then**
29: **return** $\mathcal{V}(Z)$

$\mathcal{ALC}$ axioms. Note, that 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.

For details on the formal semantics of μ-calculus and that of its combination with description logic propositions we refer to [12] and [13]. In [14], [15] it has been shown that the satisfiability of μ-calculus is an EXPTIME-complete problem.

2) Model Checking Algorithm: For a given LTS $(S, T, A, \mathcal{V})$, the semantics of a behavior constraint formula Ψ is summarized as follows:

$$\begin{aligned}
[\![\top]\!]_\mathcal{V} &= S \\
[\![\bot]\!]_\mathcal{V} &= \emptyset \\
[\![P]\!]_\mathcal{V} &= \mathcal{V}(P) \\
[\![Z]\!]_\mathcal{V} &= \mathcal{V}(Z) \\
[\![\Psi_1 \wedge_\mu \Psi_2]\!]_\mathcal{V} &= [\![\Psi_1]\!]_\mathcal{V} \cap [\![\Psi_2]\!]_\mathcal{V} \\
[\![\neg_\mu \Psi]\!]_\mathcal{V} &= S - [\![\Psi]\!]_\mathcal{V} \\
[\![\mu X.\Psi(X)]\!]_\mathcal{V} &= \bigcap \{\hat{S} \subseteq S | \hat{S} \subseteq [\![\Psi]\!]_{\mathcal{V}[X:=\hat{S}]}\} \\
[\![\langle a \rangle \Psi]\!]_\mathcal{V} &= \{s \in S | \exists (s, a, t) \in T \wedge t \in [\![\Psi]\!]_\mathcal{V}\}
\end{aligned}$$

The naive model checking algorithm is presented in Alg. 1. The procedure *evaluateFormula* computes a subset S' of the LTS states S such that all states in S' satisfy Ψ. We obtain the final boolean answer by checking whether the initial states of the LTS are in S' or not. In [16] it has been shown that the naive μ-calculus model checking for a formula of size m with alternation depth d and an LTS of size $n = |S|$ has time complexity $O(m \cdot n^d)$ under the assumption that the cost of evaluating a proposition (Alg. 1, Line 8) is negligible.

III. Offline Computable Indexes

While model checking provides a basic technique, it is not sufficient to perform model checking of each browsing process for a given query specified in the constraint specification language. Although in our scenario the complexity of queries that are mainly composed of propositions can be expected to be rather low, the naive model checking of large sets of browsing process remains very time consuming.

A. Choosing the Right Type of Model Checking Approach

In the explicit state approach, the LTS is represented extensionally using conventional data structures such as adjacency matrices and linked lists so that each state and transition is enumerated explicitly. In contrast, in the symbolic approach boolean expressions denote large LTS implicitly [17]. Typically, the data structure involved is that of Binary Decision Diagrams (BDDs) [18], which can often manipulate boolean expressions denoting large sets of states efficiently. The distinction between explicit state and symbolic representations is to a large extent an implementation issue rather than a conceptual one. BDD-based model checkers have been remarkably effective and useful for debugging and verification of hardware circuits. For reasons not well understood, BDDs are often able to exploit the regularity that is readily apparent even to the human eye in many hardware designs. Because software typically lacks this regularity, BDD-based model checking seems much less helpful for software verification.

In the monolithic approach, the entire structure of a recipe is built and represented at any time in computer memory. While conceptually simple and consistent with standard conventions for judging the complexity of graph algorithms, in practice this may be highly undesirable because the entire structure may not fit in computer memory at once. In contrast, the incremental approach (also referred to as the "on-the-fly" or "online" approach) entails building and storing only small portions of the whole structure at any time [19].

Figure 2 shows our evaluation results for performance of naive model checking implementation. In our evaluation settings, the browsing processes contained 3 ± 1 activities each, each activity had 3 ± 2 parameters, and each parameter had 3 ± 2 ontology axioms as semantic annotations (more details in Section IV). In the first case the states are loaded or modified on-the-fly. As a result, $\mathcal{ALC}$ reasoning is done during the model checking time and total query answering time increases quickly with increasing number of processes. The satisfiability problem and thus the subsumption problem in $\mathcal{ALC}$ has been shown to be EXPTIME-complete in [20]. In the second case we load all the state ontologies offline which means the expensive $\mathcal{ALC}$ inferencing done by the $\mathcal{ALC}$ reasoner at loading time before model checking. In this case, we achieve linear time complexity that verifies the theoretical time complexity of model checking (see Section II-B2). Even though browsing processes are independent of each other it is

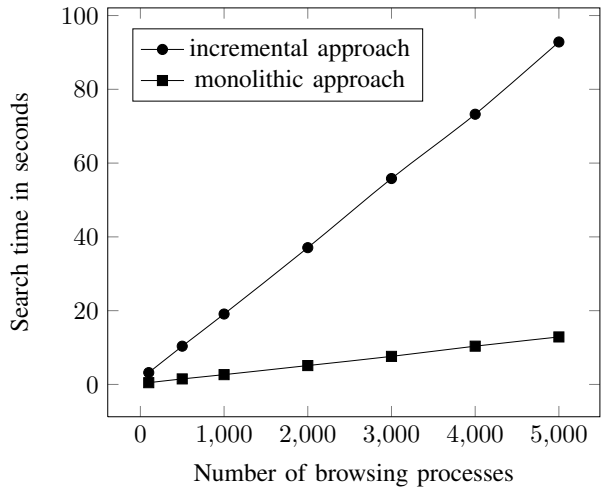

Fig. 2. Naive model checking performance.

hard to parallelize the problem because there may be mappings between the domain ontologies of the browsing processes.

Even though we achieve linear time complexity, it requires 1ms per browsing process. Considering that end users nowadays are used to extremely fast search engines, and that in a production environment we might have millions of processes, the achieved performance in the second case is still not fast enough. Developing a new model checking algorithm that has theoretically better linear time complexity in the size of LTS is out of scope of this paper. Rather, our aim is to develop indexing techniques that reduce the constant factor in the absolute time required to find matching browsing processes.

B. Proposition-States Indexes

One of the main problems of naive model checking is that it requires a lot of time for evaluating the DL propositions. This is mainly due to the following problems: (i) DL reasoning is not efficient, (ii) the naive approach checks the same proposition for the same state multiple times, and (iii) the states are checked sequentially.While optimizing DL reasoning is out of the scope of this paper, we present in the following how we address the latter two problems.

It is easy to see from Alg. 1 that it takes a lot of time for iterating over the states and checking whether they satisfy a given proposition (Lines 7-9). This takes $O(n)$ time with n the number of states in the entire LTS. In order to reduce this time, we build the proposition-states (PS) indexes. An atomic proposition is a triple $(\mathtt{s}, \mathtt{p}, \mathtt{o})$ with subject $\mathtt{s}$, object $\mathtt{o}$, and predicate $\mathtt{p}$. A PS index is a list of (P, S) pairs, where P is a proposition and S the set of states that satisfy P. The index is sorted by an ordered combination of $\mathtt{s}$, $\mathtt{p}$, and $\mathtt{o}$. This index contains only the propositions explicitly contained in the state descriptions. Therefore, it can be built offline by extracting the propositions from the states and transitions.

Alg. 2 describes how the PS indexes can be built. Whenever during the model checking of a browsing process we need to retrieve the set of states that satisfy a given proposition,

Algorithm 2: Build Proposition-State Indexes

```
Require: an LTS L = (S, T, A, λ)
  for all states s ∈ S do
    Let ns denote the namespace of s
    for all propositions p ∈ λ(s) do
      add a new row to I and let r denote this row
      insert ns:p into first column of r
      add ns:s to the entries in the second column of r
  PSO ← table I sorted by p, s, o of the first column
  POS ← table I sorted by p, o, s of the first column
  SPO ← table I sorted by s, p, o of the first column
  OPS ← table I sorted by o, p, s of the first column
```

we perform a lookup in the PS index instead of iterating over all the states of the entire model containing the LTS representations of each browsing process. This means, we replace in Alg. 1 Lines 6- 10 by the statement "**return** *lookupPS*(P)". As the PS index is sorted by proposition, such a lookup in possible in $O(\log n)$ time where n denotes the number of distinct propositions derived from the states.

Propositions $\lambda(s)$ of a state s are derived from the axioms in the state knowledge base. E.g., the second state in Figure 1 contains the axioms $\mathsf{ex{:}Track}(\mathsf{t_1})$ and $\mathsf{ex{:}partOf}(\mathsf{t_1}, \mathsf{conf})$ with named instances $\mathsf{t_1}$ and conf. Both axioms are directly added as propositions to $\lambda(s)$. If the domain knowledge of the search system provides super properties sp of the one used in an axiom, then an additional proposition $\mathsf{sp}(\mathsf{t_1}, \mathsf{conf})$ is added to $\lambda(s)$. The same applies for ontology classes and their super classes. The indexes with different orderings (PSO, POS, SPO, OPS) of predicates $\mathtt{p}$, subjects $\mathtt{s}$, objects $\mathtt{o}$ of propositions allow for queries with propositions that contain variables, e.g. $\mathsf{ex{:}partOf}(?\mathsf{x}, \mathsf{conf})$ with a variable $?\mathsf{x}$ in the subject position. Here, the POS or OPS index may be used to match predicate and object positions and retrieve matching states efficiently. Orderings SOP and OSP are not indexed mainly due to the fact that propositions with unspecified predicates (hence the trailing $\mathtt{p}$) are only relevant for rare cases in search queries.

C. Action-States Indexes

Iterating over all transitions and checking whether they satisfy the constraints of a requested action (Alg. 1, Lines 17-19) similarly consumes a lot of time. It takes $O(n)$ time with n the number of transition in the entire LTS. In order to save time for verifying actions, we build the action-states (AS) indexes. This index is a list of (A, S^2) pairs, where A is an action, and S^2 the set of states pairs that are connected via the action A. The index contains only the actions that are explicitly mentioned in the transitions of the LTS and can be built offline.

Alg. 3 describes how the AS indexes are built. Actions are characterized by the action type $\mathtt{t}$ (input, output, or computational), communication channel $\mathtt{c}$, and parameters $\mathtt{p}$. An action further describes constraints (propositions) over the parameters, which are verified by means of the PS indexes. An activity matches if the state s_2 satisfies the propositions of an input or a computational action. Since output actions do not change the state knowledge, s_1 and s_2 can be used likewise to

Algorithm 3: Build Action-States Indexes

Require: an LTS $L = (S, T, A, \lambda)$
  for all transition $t = (s_1, a, s_2) \in T$ **do**
    Let *ns* denote the namespace of s_1
    add a new row to AT and let r denote this row
    insert ns:a into first column or r
    add (ns:s_1, ns:s_2) to the entries in the 2. column of r
  TCP $\leftarrow$ table I sorted by $\mathtt{t}, \mathtt{c}, \mathtt{p}$ of the first column
  CTP $\leftarrow$ table I sorted by $\mathtt{c}, \mathtt{t}, \mathtt{p}$ of the first column

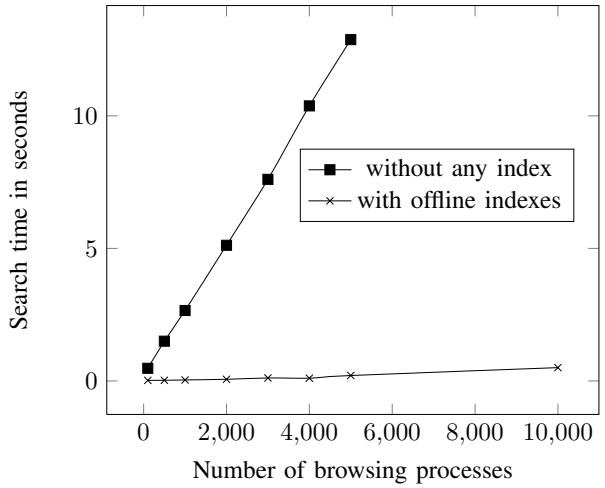

Fig. 3. Performance with and without offline computable indexes

verify the propositions. The action parameters are necessary to ensure that the requested propositions hold for the action parameters and not for other process resources. Hence p trails the entries of the first column.

We replace in Alg. 1 Lines 16-20 by the statement "**return** *lookupAS*(a)". Since the indexes are sorted by action types and channels, such a lookup is possible in $O(\log n)$ time where n is the number of transitions in the LTS.

Figure 3 shows the performance of the so obtained search approach. The query answering time is tremendously reduced. E.g., the query answering time is reduced by the factor 63 for 5000 browsing process descriptions (from 12.8 seconds to 0.205 seconds). Even for 20000 processes, the indexes based approach returns matching processes in less than 1.2 seconds.

IV. Implementation

We developed APIs for recipe descriptions and requests. Processes and their LTS representations are serialized in form of RDFS statements based on appropriate RDFS vocabularies we developed. LTS representations and domain ontologies used by processes are stored in the OWLIM-SE semantic repository that provides a SPARQL query interface. SPARQL is the language used to query the repository of LTS. OWLIM supports RDFS reasoning and allows to reason over heterogeneous LTS descriptions based on the domain ontologies.

Search queries are decomposed by our μ-calculus reasoner developed as a Java component that is flexible enough to provide the incremental, monolithic, and indexes based model checking capabilities. The μ-calculus reasoner is placed on top of the locally installed OWLIM-based LTS repository. During the experiments we allocated 1GB of main memory for OWLIM including its internal index structures. The index structures introduced in the present work are implemented as Java data structures (TreeMap) within the μ-calculus reasoner.

In the evaluation experiments shown above, we examined the search performance by means of measuring the query answering time with different model checking approaches. We did not experience any shortage of main memory while we conducted the experiments on commodity hardware with an Intel Core2Duo 2.6GHz CPU and 4GB main memory.

Test Data: Existing web browsing processes from the IBM CoScripter repository are the basis of the large test collection used in our experiments. The CoScripter repository currently hosts over 6000 web automation scripts. In our analysis of CoScripts we observed that many end user browsing processes are rather short comprising a small number of activities performed sequentially [21]. The reason is that many existing CoScripts automate interactions with only one website. More precisely, the processes in our experiments can be characterized by 3 ± 1 input/output activities in average with about 3 ± 2 parameters per activity and 3 ± 2 ontology axioms describing each parameter.

Based on the analysis of the CoScripts, we generated semantic browsing process descriptions. Details are provided at http://people.aifb.kit.edu/mju/recipe. The correctness wrt. to content of existing CoScripts cannot be guaranteed, as we do not focus on automatic learning of browsing processes. Still, we can argue that the test data complexity corresponds to the browsing process complexity of the scripts. In average, the search queries of the experiments are conjunctive and disjunctive compositions of 4 proposition and one action existential queries that eventually occur in desired recipes. Each μ-calculus proposition describes 2 instances and further 3 DL axioms in average. E.g., the following proposition P of a search request contains two class membership axioms and an object property specifying the relation between the two instances (here, ?x denotes a variable).

$$P \equiv \mathsf{ex{:}Chair}(?\mathsf{x}), \mathsf{ex{:}chairs}(?\mathsf{x}, \mathsf{icws}), \mathsf{ex{:}Conference}(\mathsf{icws})$$

An evaluated request is Ψ, where $P_1, \ldots, P_4$ denote propositions with complexity similar to the complexity of P.

$$\Psi \equiv (P_1 \vee_\mu P_2) \wedge_\mu (\mathbf{eventually}\ P_3 \vee_\mu \mathbf{eventually}\ P_4)$$

Domain ontologies used to describe the resources in requests belong to the same set of RDFS ontologies used for the description of browsing processes. We used 4 public ontologies such as the Semantic Web Research Community ontology. The largest ontology contains 70 classes, and 48 object properties.

V. Related Work

The Semantic Web has proposed the annotation of web pages in order to describe the information content. Apart from the fact that still most of the web pages are not annotated, it is hard to build a server-sided semantic information search

engines since a crawler will be unable to reach and index the semantic annotations within deep web pages. Linked Data separates the structured data from the traditional Web (and as a result also from the end users) completely. The Linked Data approach is primarily useful for application developers since end users cannot be expected to consume RDF directly. That is, end users still require human understandable applications to interact with, and furthermore the providers would keep on protecting their valuable data by controlling access, e.g. with the help of user interaction elements such as web forms.

Semantic search over RDF and Linked Data as in [22] enables querying Linked Data by traversing the web of Linked Data. The completeness of query answering over Linked Data has been studied in [23]. Even if Linked Data query approaches were extended to support dynamic data, e.g. by integrating so-called Linked Data Services [24], the suitability of the approaches will remain limited for the publicly available free data only. That is, Semantic Search approaches (i) heavily rely on the availability of structured data, and (ii) providers are expected to provide access to their data through APIs. But, many providers do not follow this, and (iii) even if access to data is provided, the data is usually allowed to be used for advertisement purposes only. Also, the data of one provider is often not semantically aligned with data from other providers.

The work "Navigational Plans For Data Integration" by Friedman et al. [25] is related to our approach wrt. information need driven search for web scripts. As the navigational paths are stored and returned as search results, this approach can deal with dynamic information published within web pages. It presents a sound and complete algorithm for computing navigational paths for a given information query and set of source descriptions. Computed navigational paths are however subsets of the web graph only. The algorithm cannot compute paths that consist of data flow between web pages, which are not connected in the web graph. Consider our running example with a web page A listing the track chairs, and another page B listing publications of a given author. Assume that A is not linking to B. For a query for publications of track chairs, the algorithm presented in [25] is not able to compute the composition of outputs of A and inputs of B. Further, the proposed use of a mediated schema is hardly applicable to the Web. It is also not possible to define temporal constraints over navigational plans that answer the information need.

Search engine provider collect user's click trails as well as the data they fill in the forms, e.g. with toolbar-like browser plugins. There exists a plethora of work, e.g. [4], [26] on predicting next step or the target web page by analyzing click trails. Click trails can be seen as simple browsing processes as they are sequences without variables and without data flow. The major difference is that click trail analysis aims at helping users to find the relevant pages faster mostly based on syntactic analysis of pages that the user has visited in the current session. But users still have to know which pages are relevant and have to figure out what to do with which pages once they have been found. In contrast, in our approach each browsing process has a purpose, and we aim at finding the appropriate browsing processes for the current need. Another difference is in the acceptance of the underlying technology. End users often do not see any direct added value of a toolbar plugin, which is a reason why toolbars are often delivered as part of some other software. In our approach end users have direct incentives for sharing their browsing processes as well as full control on whether and with whom they share these.

Existing μ-calculus model checkers such as nuSMV (http://nusmv.fbk.eu) typically support symbolic model checking based on BDD and/or SAT. To the best of our knowledge there is no ready to use μ-calculus reasoner based on explicit state representation. The $\mu\mathcal{ALCQ}$ logic introduced in [27] extends $\mathcal{ALC}$ by fixpoint constructs and qualified number restrictions. It has been shown in [27] that $\mu\mathcal{ALC}$ is equivalent to μ-calculus. However, there does not exist any implementation of a $\mu\mathcal{ALC}$ reasoner that we could have used. Our implementation of a μ-calculus model checker was also necessary in order to have a common platform for comparing different model checking approaches.

An index over partially ordered complex behavioral constraints for the classification of complex services was introduced in [21]. Constraints are mapped to entire services as opposed to a mapping to states used in the present approach. Both approaches are complementary as (i) behavior classes from [21] provide another offline index for complex constraints and (ii) our indexing structures allow to compute class memberships more efficiently and are still required for the verification of any constraints that were not covered by the service classification.

A metric to quantify process similarity based on behavioral profiles [28], which is grounded in the Jaccard coefficient, leverages behavioral relations between process model activities. The metric is successfully evaluated towards its approximation of human similarity assessment. So far, we did not consider similarity of browsing processes. In general, if a process p simulates another process q and it is known that p is a match for a formula ϕ, then q can be directly added to the set of matches for ϕ. Incorporating such an index may bring further search performance gains.

VI. Conclusion and Outlook

Sophisticated use cases have complex information needs. Information needs that require information from various websites are not served satisfactorily by the state of the art search engines. This leaves end users spending a lot a time for searching and compiling together required information from various web pages. In this paper, we targeted this problem from a completely new perspective. We build on a bottom-up approach that proposes capturing and sharing of end user browsing processes [29], as opposed to the top-down approach that requires annotated websites in the first place.

When the number of such web browsing processes increases, efficient techniques for finding browsing processes suitable for an information need will be required. In this paper, we have presented an efficient logic-based technique for searching end user browsing processes. Our main focus

was to develop an efficient search technique based on known model checking techniques. We presented offline indexes for propositions and activities on top of the underlying model checking algorithm to achieve a scalable and efficient search for browsing processes. Regarding offline computable indexes, we have shown that indexing proposition and the states in which they hold as well as indexing which actions are possible in which states brings significant performance gains.

Currently, we are developing an online index for complex constraint formulas as an additional indexing structure enhancing the presented work. In comparison to the behavior constraints of service classes that are supposed to be given [21], our online index will automatically collect complex constraints from end user posed queries during runtime. Since the number of such complex queries is potentially infinite we plan to incorporate a well-known randomized algorithm for deciding which answers should be kept in the index. The randomized algorithm is $2H_k$-competitive for k pages in the online cache, which means that for any sequence of queries it requires at most $2H_k$ (H_k denotes the k-th harmonic number) more time than an optimal algorithm for serving the sequence of queries [30]. We are developing a lookup procedure for efficiently searching subformulas in the online index, which promises to bring further performance gains.

In future, we wish to continue this work to support (i) composition of browsing processes, and (ii) ranking of browsing processes also including the social dimension as in [31]. The ranking can be employed for a top-k search technique for browsing processes and composition that delivers results more promptly while compromising on the completeness of the search results.

ACKNOWLEDGMENT

The authors acknowledge the support of the European Community's Seventh Framework Programme FP7-ICT-2011-7 (XLike, Grant 288342).

REFERENCES

[1] M. K. Bergman, "The deep web: Surfacing hidden value," *The Journal of Electronic Publishing*, vol. 7, no. 1, 2001.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web: a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities," *Sci. Am.*, vol. 5, no. 284, pp. 34–43, 2001.

[3] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data - the story so far," *Int. Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009.

[4] A. Singla, R. White, and J. Huang, "Studying trailfinding algorithms for enhanced web search," in *SIGIR*, F. Crestani, S. Marchand-Maillet, H.-H. Chen, E. N. Efthimiadis, and J. Savoy, Eds. ACM, 2010, pp. 443–450.

[5] R. W. White and J. Huang, "Assessing the scenic route: measuring the value of search trails in web logs," in *SIGIR*, F. Crestani, S. Marchand-Maillet, H.-H. Chen, E. N. Efthimiadis, and J. Savoy, Eds. ACM, 2010, pp. 587–594.

[6] J. Teevan, C. Alvarado, M. S. Ackerman, and D. R. Karger, "The perfect search engine is not enough: a study of orienteering behavior in directed search," in *CHI*, E. Dykstra-Erickson and M. Tscheligi, Eds. ACM, 2004, pp. 415–422.

[7] 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.

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

[9] 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.

[10] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Y. Halevy, "Google's deep web crawl," *PVLDB*, vol. 1, no. 2, pp. 1241–1252, 2008.

[11] F. van Harmelen, V. Lifschitz, and B. Porter, Eds., *Handbook of Knowledge Representation*, 1st ed., ser. Foundations of Artificial Intelligence. Elsevier Science, 2008.

[12] C. Stirling, *Modal and Temporal Properties of Processes*. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[13] S. Agarwal, S. Lamparter, and R. Studer, "Making web services tradable: A policy-based approach for specifying preferences on web service properties," *J. Web Sem.*, vol. 7, no. 1, pp. 11–20, 2009.

[14] R. S. Streett and E. A. Emerson, "The propositional mu-calculus is elementary," in *ICALP*, ser. LNCS, J. Paredaens, Ed., vol. 172. Springer, 1984, pp. 465–472.

[15] S. Safra, "On the complexity of omega-automata," in *FOCS*. IEEE Computer Society, 1988, pp. 319–327.

[16] E. A. Emerson and C.-L. Lei, "Efficient model checking in fragments of the propositional mu-calculus (extended abstract)," in *LICS*. IEEE Computer Society, 1986, pp. 267–278.

[17] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Inf. Comput.*, vol. 98, no. 2, pp. 142–170, 1992.

[18] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[19] C. Jard and T. Jéron, "On-line model checking for finite linear temporal logic specifications," in *Automatic Verification Methods for Finite State Systems*, ser. LNCS, J. Sifakis, Ed., vol. 407. Springer, 1989, pp. 189–196.

[20] F. M. Donini and F. Massacci, "EXPTIME tableaux for $\mathcal{ALC}$," *Artificial Intelligence*, vol. 124, no. 1, pp. 87–138, 2000.

[21] M. Junghans, S. Agarwal, and R. Studer, "Behavior classes for specification and search of complex services and processes," in *ICWS*, C. A. Goble, P. P. Chen, and J. Zhang, Eds. IEEE, 2012, pp. 343–350.

[22] G. Ladwig and T. Tran, "Linked data query processing strategies," in *International Semantic Web Conference (1)*, ser. LNCS, P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, and B. Glimm, Eds., vol. 6496. Springer, 2010, pp. 453–469.

[23] A. Harth and S. Speiser, "On completeness classes for query evaluation on linked data," in *AAAI*, J. Hoffmann and B. Selman, Eds. AAAI Press, 2012.

[24] S. Speiser and A. Harth, "Integrating linked data and services with linked data services," in *ESWC (1)*, ser. LNCS, G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, Eds., vol. 6643. Springer, 2011, pp. 170–184.

[25] M. Friedman, A. Y. Levy, and T. D. Millstein, "Navigational plans for data integration," in *AAAI/IAAI*, J. Hendler and D. Subramanian, Eds. AAAI Press / The MIT Press, 1999, pp. 67–73.

[26] M. Bilenko and R. W. White, "Mining the search trails of surfing crowds: identifying relevant websites from user activity," in *WWW*, J. Huai, R. Chen, H.-W. Hon, Y. Liu, W.-Y. Ma, A. Tomkins, and X. Zhang, Eds. ACM, 2008, pp. 51–60.

[27] G. D. Giacomo and M. Lenzerini, "A uniform framework for concept definitions in description logics," *Journal of Artificial Intelligence Research (JAIR)*, vol. 6, pp. 87–110, 1997.

[28] M. Kunze, M. Weidlich, and M. Weske, "Behavioral similarity - a proper metric," in *BPM*, ser. Lecture Notes in Computer Science, S. Rinderle-Ma, F. Toumani, and K. Wolf, Eds., vol. 6896. Springer, 2011, pp. 166–181.

[29] S. Agarwal and C. J. Petrie, "An alternative to the top-down semantic web of services," *IEEE Internet Computing*, vol. 16, no. 5, pp. 94–97, 2012.

[30] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive paging algorithms," *J. Algorithms*, vol. 12, pp. 685–699, 1991.

[31] B. Bahmani and A. Goel, "Partitioned multi-indexing: bringing order to social search," in *Proc. of the 21st Int. Conf. on World Wide Web*, ser. WWW '12. New York, NY, USA: ACM, 2012, pp. 399–408.