Swapping out Coordination of Web Processes to the Web Browser

Sudhir Agarwal
Karlsruhe Service Research Institute (KSRI)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
sudhir.agarwal@kit.edu

Martin Junghans
Karlsruhe Service Research Institute (KSRI)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
martin.junghans@kit.edu

***Abstract*—Consumption of (business) processes provided in form of Web sites have become a part of our daily life for attending our personal and business needs. In order to obtain the best solution for a particular task, users often combine several Web processes. However, the coordination of the execution of such Web process compositions is completely manual demanding the user to enter same or logical dependent data multiple times. We argue that a part of such coordination effort could be automatized by swapping out the uncreative coordination tasks to the Web browser. Our solution allows users to compose Web processes as generic solutions and execute the compositions with appropriate parameters every time they need to perform a concrete task, thus relieving them from a lot of manual coordination effort. We show how such Web process compositions can be formalized, obtained and executed inside a common Web browser with automatic flow of data among different parties despite heterogeneous data.**

I. Introduction

Most of the interesting (business) processes need to interact with the user multiple times during their execution, e.g. for obtaining inputs, providing outputs or resolving non-determinism in order to proceed with further execution. In order to interact with the user elements for displaying information as well as elements for receiving user input are needed. In the Web, such multi-step, multi-interactive, non-deterministic processes are implemented as Web sites, while single step, deterministic utility procedures are often offered as Web services. Web sites build the much larger part of the Web than the atomic Web services[1]. Web sites offer processes, while Web services mainly offer simple utility procedures, e.g. for conversion of formats, currencies or querying a database etc. In the rest of the paper, we use the term *Web Processes* for RPC based Web services, RESTful Web services, Web sites etc.

End users use Web sites for accomplishing their simple day to day tasks as well as complex business needs. Users often need more than one Web process to accomplish a task at hand because of reasons like (1) users wish to compare the outcomes of different Web processes and select the best one, and (2) complex tasks that can not be performed completely with one Web process or (3) when a process needs inputs that a user obtains as outputs of other processes, to name a few.

[1]around 30,000 publicly available WSDLs according to seekda [1] vs. a few billion Web sites even without considering dynamic Web sites.

Example Scenario: Consider Mary who is a secretary and needs to arrange travel for her boss very often. Every time, she is supposed to plan a trip for her boss, she needs to search and book the most suitable flight, hotel and rental car. For doing so she uses a bunch of Web sites. For flight booking sites, she need to enter date and time considering the timetable of her boss multiple times, check the flight availability and compare the prices etc. Furthermore, she needs to check the availability of the hotels that are not too far away both from the location of the meeting her boss want to attend as well as the airport. Especially, in case the meeting location is far from hotel, she needs to find a rental car of appropriate class for reasonable price and availability in the duration of the stay of her boss.

Currently, users have to coordinate the execution of various Web sites manually, e.g. by manually entering same (or logical dependent) data in different forms multiple times, trying out different input values, aggregate results of various Web processes. Considering that many tasks that the users accomplish with the help of multiple Web processes need to performed again and again (e.g. travel booking as described in Example I), supporting a user with automatic techniques in coordinating the Web processes can save a lot of human effort.

In the recent years, many techniques have been developed with the aim of providing users with support for automation while working in the Web. The initial approaches e.g. [2] targeted mainly the data on static Web pages. Later the idea of semantic description of Web data has been applied for Web services resulting in approaches like OWL-S [3] and WSMO [4]. Automatic composition techniques for semantic Web services, e.g. [5] have considered RPC style Web services, even though the mentioned semantic Web service description techniques provide with models for describing composite Web services as well. The execution environments like OWL-S Virtual Machine [3], [6] and Semantic Execution Environment [7] focus mainly on the execution of workflows that have semantic Web services as atomic activities. To the best of our knowledge, the composite service description techniques have not been applied for describing dynamics of Web sites, nor the composition algorithms have been developed for the composition of various Web sites, nor there are any semantic execution environments that support a user in executing composed Web sites in the Web browser.

In this paper, we present an approach which supports users in the accomplishment of recurring tasks in the Web. The central idea is to allow users to define solution templates as complex decentralized workflow with many Web processes and store them as "intelligent" bookmarks. For every concrete instantiation of a problem appropriate bookmark can be selected triggering the execution of the complex underlying workflow. For defining such Web processes, we need a process description language, which we briefly introduce in Section II-A. In order to be able to define the solution templates declaratively, the descriptions of many Web processes should be available. In Section II-B we give an overview of our semi-automatic techniques for obtaining descriptions of the processes implicit in the flow of Web pages. In order to be able to search and rank the process descriptions in a large pool of process descriptions, there is a need for appropriate search and ranking mechanisms. We introduce them in Section II-C and Section II-D respectively. Having all the preliminaries introduced, we develop in Section III an automatic technique for supporting users in the task of defining the solution templates. In Section IV we present the overall architecture of our system with implementation details of our Web browser based graphical solution template synthesis and execution prototype. We conclude in Section VI after discussing related work in Section V.

II. Preliminaries

In this section we give short overviews of some existing technologies which are needed to develop the main contribution of the paper. Automatic techniques for generating appropriate compositions of Web processes are useful only if there is a large pool of semantic descriptions of Web processes available. In II-A, we give an overview of the process description language that we use for describing the dynamics of Web sites. In II-B, we give a brief introduction of our view of Web sites as processes as well as an overview of our semi-automatic approach for obtaining descriptions of processes implicit in the flow of Web pages.

A. Semantic Description of Web Sites as Processes

In this section, we present an overview of the suprime Process Description Language (*suprimePDL*) that we use to describe the information flow and control flow among the Web pages. *suprimePDL* is based on the π-calculus process algebra. For details on the syntax and formal semantics of the language, we refer to [8], [9].

In *suprimePDL* an agent is defined with $A(x_1, \ldots, x_n) \stackrel{def}{=} P$, where $x_1, \ldots, x_n$ are the only names that may occur then unbound in the process expression P which is defined recursively as follows:

$$\begin{aligned} P \quad ::= \quad & \mathbf{0} \mid y[v_1 \ldots, v_n].Q \mid y\langle x_1 \ldots, x_n\rangle.Q \mid \\ & l(x_1, \ldots, x_n)(y_1, \ldots, y_m).Q \mid \\ & [\omega]Q \mid P_1 \parallel \ldots \parallel P_n \mid P_1 + \ldots + P_n \mid \\ & @A\{y_1, \ldots, y_n\} \end{aligned}$$

Element	Maps to
Base URL of Web page / Link	Logical URI of ontology
Display element id	Ontology class
Content of a display element	Ontology instance of the class corr. to the display element id
Variable name of link	Ontology class
Variable value of link	Ontology instance of class corr. to the variable
Form name	Complex ontology class
Form field id	Property of the class corresponding to the form name
Form field name	ontology class representing the range of the property corresponding to the field id
Form Field Value	Instance

Table I
CORRESPONDENCE BETWEEN PAGE CONTENT AND ONTOLOGY

The *Null process* $\mathbf{0}$ denotes a process that performs no action and is often used as termination symbol. *Input process* $y[v_1, \ldots, v_n].Q$ denotes a process that takes inputs at the connection y and binds them to names $v_1 \ldots, v_n$. The subsequent behavior of this process is defined in the process expression Q. *Output process* $y\langle x_1, \ldots, x_n\rangle.Q$ denotes a process that outputs the names $x_1 \ldots, x_n$ at connection y. Its subsequent behavior is defined in the following expression Q. *Local process* $l(x_1, \ldots, x_n)(y_1, \ldots, y_m).Q$ is a process that performs the operation l with the arguments $x_1, \ldots, x_n$ and produces output $y_1, \ldots, y_m$. Its subsequent behavior is defined in the process expression Q. *Conditional Process* $[\omega]Q$ denotes a process which behaves like Q if the condition ω is true, otherwise it behaves like $\mathbf{0}$. *Composition* $P_1 \parallel \ldots \parallel P_n$ denotes the parallel composition of processes $P_1, \ldots, P_n$. *Choice* $P_1 + \ldots + P_n$ denotes a process with n alternatives $P_1, \ldots, P_n$, from which only one is selected for further execution. *Agent Invocation* $@A\{y_1 \ldots, y_n\}$ denotes invocation of a defined agent identifier A. Furthermore, an agent may have a set of semantic SPKI certificates as proof of values of some non-functional properties.

B. View of Web Sites as suprimePDL Processes

Table I summarizes the correspondence between the static part of a Web page and the ontology elements. For each new Web site, we create an ontology with the logical URI derived from the base URL of the Web site. In the semantic description of the content of a link, the arguments of a link are modeled as classes in the ontology, and the values of the arguments as instances of the classes corresponding to the arguments. An HTML form corresponds to a complex ontology class in the ontology. The names of the input elements of the form are the properties of the complex class representing the whole form. The range of a property corresponding to an input element is modeled as an ontology class. The name of the class 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 can be selected. In these cases, the provided values are modeled as ontology

Web Artifact	Element of the Process Description Language
URL	Agent identifier
Web page	Composition of a set of outputs and a choice from a set of links and forms
Selection of a link	Invocation of an agent identifier
Submission of a form	Input process
CGI script	Execution of a local Operation
Web	Agent identifier composed of concurrently running Web pages

Table II
MAPPING BETWEEN WEB ARTIFACTS AND ELEMENTS OF THE FORMALISM

instances, while the class representing the range of an input field as enumeration class instead of a normal class. Thus, we obtain an ontology with classes, instances and relationships for each Web site. Automatic techniques for detecting mappings and alignments, e.g. [11] in such a large pool of ontologies are necessary.

A set of mappings, illustrated in Table II, is defined between the Web artifacts and the elements of our process description language. In our view, a URL is equivalent to an agent identifier, whereas the selection of a link, which is a usage of a URL, is equivalent to invocation of an agent identifier with concrete values for the arguments. In our model, a Web page corresponds to a process which is a composition of a set of outputs and a choice from a set of links and forms. Formally, a Web page P that display l values $x_1, \ldots, x_l$, contains m links $u_1, \ldots, u_l$ and n forms $f_1, \ldots, f_n$ can be described as follows:

$$y\langle o_1, \ldots, o_l\rangle.\mathbf{0} \parallel \{U_1 + \ldots + U_l + F_1.N_1 + \ldots + F_n.N_n\},$$

where $U_1, \ldots, U_l$ denote the URLs that the links $u_1, \ldots, u_l$ are respective invocations of and $F_1.N_1, \ldots, F_n.N_n$ the input processes corresponding to the forms $f_1, \ldots, f_n$.

Our semi-automatic acquisition of semantic process descriptions of Web sites automatically crawl the (dynamic) Web pages and create ontologies for the terms occurring on a Web page, especially in the links and forms as well as description of the process implicit in the flow of crawled Web pages. Such automatically created ontologies and process descriptions can be further refined manually with a browser based graphical editor for *suprimePDL* [12], [13].

C. Search

For the synthesis of a coordinating process, it is required that the processes that need to be coordinated are known. Furthermore, during the synthesis, it is required to find those processes that can be glued together to a given process. Technically it means, that for a given interaction pattern, processes with inverse interaction pattern need to be detected automatically. In [14], we have developed a technique for finding processes that fulfill constraints on the functionality, including temporal constraints (desired order of activities). The query formalism for constraints is a combination of the μ-calculus temporal logic and $\mathcal{SHIQ}(\mathbf{D})$ description logic. In [14], we have developed a model checking algorithm that checks for a given query and a given process description in *suprimePDL*, whether the process description fulfills the query or not. We will use the model checking technique developed for the purpose of finding processes with required temporal structure and functionality from within our synthesis algorithm presented in the next section.

D. Ranking

Especially, when the set of results for a given query can be large, there is need for ranking the results. In this section we give an overview of the ranking approach that will be used to rank the automatically synthesized solution templates as well the outputs returned as the result of a process consumption. In addition, the results of the process search component can be ranking using the same approach. Since, user preferences on NFPs are often non-crisp and computation of optimal solution is a computationally hard, we have developed a ranking technique based on Fuzzy inferencing. Users express the preferences as Fuzzy If-Then rules with the help of given customizable vocabulary instead of dealing with concrete numerical values and weights [15]. The computation of the rank for given user preferences as Fuzzy If-Then rules is carried out as follows: (1) The concrete values of NFPs of a process are fuzzified (2) For every rule the degree of fulfillment of the premise is computed and the part of the fuzzy set in the conclusion that is above the degree is chopped away. (3) All the chopped fuzzy sets (one per rule) are aggregated to one fuzzy set which represents the overall score of the process as a fuzzy set. (4) The fuzzy set is defuzzified to obtain a crisp value between 0 and 1, which represents the acceptance score of the process. After performing the above 4 steps for each process, we obtain a score for each process by which they can be sorted. For further details of the ranking approach, we refer to [16].

III. SYNTHESIS OF SOLUTION TEMPLATES

In this section, we present an automatic technique to synthesize solution templates. Given the logical constraints on the information flow among processes, a solution template also determines the sequences in which processes are invoked. Different independently acting Web processes invoked by a user do not communicate with directly with each other, but rather via the user, e.g. when the outputs of one Web process need to fed to another Web process. As a consequence, the problem of automating the coordination can be seen as the task of synthesizing a controlling process C that runs in the user's Web browser. Our approach allows user to specify constrains on the controlling process with respect to desired control and information flow and constructs controlling processes that fulfill user's constraints by combining the semantic descriptions of the appropriate Web processes available in the process repository created with the semi-automatic acquisition technique [13]. In Section III-A, we show how the a controlling process can be modeled with *suprimePDL*.

Since, modeling solution templates can be a very tedious task, if performed manually, we develop an algorithm for synthesizing such solution templates automatically in Section III-B.

A. Modeling Controlling Processes

In this section we show how such a controlling process can be modeled with our process formalism semantically. When a user models a controlling process, the user has certain constraints regarding the data flow, the control flow, the properties of the data as well changes made by the whole process. We derive three requisites from the above mentioned three reasons for the usage of a controlling process. (1) When no single process provides all desired outputs, a parallelization of component processes is required in order to simulate the possible data flow among them. Also, when there are several similar processes, i.e. processes that provide the same results, the parallelization can collect results of all processes. (2) When the number of inputs required from the user can be reduced by reusing known inputs that are provided by the user or by previously obtained outputs of other processes, then we need to model the data flow between the controlling process and the process for which inputs can be provided. As the controlling process is the invoker, it always needs to accept outputs provided by the processes. (3) When the output of the composed process needs to be the best among all alternatives, then the resulting information obtained from all component processes shall be aggregated.

Suppose the controlling process is denoted by C and the Web processes that are supposed to be composed together with the help of the controlling process C are denoted by $P_1, \ldots, P_n$. Then a solution template is denoted by $C \parallel P_1 \parallel \ldots \parallel P_n$. That is, all the Web processes run in parallel to each other and also to the controlling process.

Modeling Data Flow On the level of formalism it means that for every input activity a in a P_i if there is an output activity b such that a is connected to b then b is in C. Similarly, for every output activity b in a P_i if there is an input activity a such that b is connected to a then a is in C. Unconnected input and output activities in a P_i imply that the input values will be provided by the user and output values will be sent to the user respectively. If a user is willing to provide some inputs, then there will be corresponding (unconnected) input activities in C. Similarly, if a user wishes to obtain some output from C, there will be corresponding (unconnected) output activities in C.

A data flow connection from an output activity to an input activity is valid only if the values emitted by the output activity are of the types expected by the input activity. Recall from Section II-A that we model the data objects as instances of ontological classes as well as the classes along with their relationships with other classes as domain ontology. Having such semantic descriptions, such a type checking can be achieved by standard ontology reasoning procedures for classification despite heterogeneity in the schemata of the processes P_i and P_j.

Local Operations Often output values provided by a process can not be fed into the input of another process directly. For example, when values from different sources (processes) first need to aggregated and only the aggregated value and not the individual values are used as the input of another process. For such calculation purposes, we assume a set of operations that can be performed locally respective to the controlling process. Whenever a preprocessing of data is needed, one or more local operations are added to the definition of the controlling process C at the appropriate position (refer to local operations in Section II-A).

B. Automatic Synthesis of Solution Templates

The synthesis algorithm computes a set of solution templates for a given problem, which is described by a user's requirements. The user formulates the desired properties of a solution template as a tuple $(\mathcal{I}, \mathcal{O}, \Gamma, \Pi^o, \Pi^t)$ that contains a set $\mathcal{I}$ of inputs, a set $\mathcal{O}$ of outputs, a description Γ of constraints on the outputs and how these outputs are related to the inputs, and sets Π^o and Π^t of preferences on outputs and templates, respectively. The algorithm not just composes several processes such that a template provides the required outputs taking requirements and preferences into account. It focuses on synthesizing a controlling process that eases controlling the process execution by automating non-challenging tasks like provision of inputs or forwarding parameters.

The set $\mathcal{I}$ of inputs describes inputs that a user is willing to provide to processes of a template. In our example, Mary wants to provide $\mathcal{I} = \{\mathsf{Time(flightArrivalTime)}, \mathsf{Airport(flightStart)}, \mathsf{Airport(flightEnd)}, \mathsf{Time(carPickupTime)}, \mathsf{City(carPickup)}, \ldots, \mathsf{CreditCard(cc)}^*, \mathsf{UserProfile(user)}^*\}$. The last two inputs cc and user are marked for manual submission. The execution engine will not perform any submit action if one of these marked inputs occurs in the form.

The desired outputs $\mathcal{O}$ must be provided by a solution template. For instance, Mary requires a Web process that delivers basic information like start and end location and/or time for flight, rental car, and hotel, as well as pricing information and perhaps payment details. She could serialize this aspect as $\mathcal{O} = \{\mathsf{FlightTicket(ft)}, \mathsf{Time(ftEndTime)}, \mathsf{Price(fp)}, \mathsf{RentalContract(car)}, \mathsf{Price(cp)}, \ldots\}$.

Γ denotes a logical expression that describes the relation between inputs and outputs of the composed process. It allows also to constrain the data flow among several Web processes and to constrain the outputs of the processes. For example, $\Gamma = \{\mathsf{equiv(flightEnd, ftEnd)}, \mathsf{before(ftEndTime, carPickupTime)}, \leq \mathsf{(fp, 100)}, \ldots\}$ is a set of constraints that are interpreted as a conjunctive query. Each constraint can be a binary or unary predicate. $\mathsf{equiv(flightEnd, ftEnd)}$ states that the desired destination equals the destination airport of the flight ticket. $\mathsf{before(ftEndTime, carPickupTime)}$ relates the dependency between the arrival time and the pickup time of the rental car. It allows to use the outputs

of the flight booking process for a subsequent rental car arrangement process (if it is required to combine these two different processes). Web process outputs are constrained (filtered) when a parameter is compared to a literal, as in $\leq (\mathsf{fp}, 100)$. It is also possible to filter based on a variable value instead of specifying a constant value at design time. Then, the execution will generate an input form for that variable value. Summarizing, users only need to deal with outputs satisfying the constraints Γ. Thus, Γ can be regarded as a set of filters on processes as process execution only continues if the outputs fulfill the constraints.

Finally, a set Π^o of preferences on the outputs and a set of preferences Π^t on composed processes can be specified as further requirements. Here, Mary prefers offers (obtained at execution time) with $\Pi^o = IF\ (\mathsf{fp} = low \wedge \mathsf{travelDuration} = short)\ THEN\ \mathsf{acceptance} = excellent$, whereas travelDuration denotes the overall duration carPickupTime−flightDepartureTime of the travel. The preferences Π^t are analogously expressed using Fuzzy If-Then rules as mentioned in Section II-D. Preferences on a template comprise non-functional properties of a process like availability or user rating and determine a ranking of the solution templates.

Algorithm 1 searchProcess

Require: Desired properties $(\mathcal{I}, \mathcal{O}, \Gamma, \Pi^o, \Pi^t)$, $\mathcal{P} \leftarrow \{\}$
  $\mathcal{P}_{match} \leftarrow$ *match*$(\mathcal{I}, \mathcal{O}, \Gamma)$
  for all $P \in \mathcal{P}_{match}$ **do**
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{P || $*synthContrProc*$(P, C, \Gamma)\}$
  for all $\mathcal{O}_1 \in 2^{\mathcal{O}}$ **do**
5: **if** $|\mathcal{O}_1| > 0$ and $|\mathcal{O}_1| < |\mathcal{O}|$ **then**
      $\mathcal{P}_1 \leftarrow$ *searchProcess*$(\mathcal{I}, \mathcal{O}_1, \Gamma, \Pi^o)$
      $\mathcal{P}_2 \leftarrow$ *searchProcess*$(\mathcal{I}, \mathcal{O} - \mathcal{O}_1, \Gamma, \Pi^o)$
      for all $P_1 \in \mathcal{P}_1$ and $P_2 \in \mathcal{P}_2$ **do**
        $\mathcal{P} \leftarrow \mathcal{P} \cup \{P_1 || P_2 ||$ *synthContrProc*$(P_1 || P_2, C, \Gamma)\}$
10: $\mathcal{P} \leftarrow$ *addResultRanking*(*addProcessFilter*$(\mathcal{P}, \Gamma), \Pi^o)$
  return *rankTemplates*$(\mathcal{P}, \Pi^t)$

Given desired properties $(\mathcal{I}, \mathcal{O}, \Gamma, \Pi^o, \Pi^t)$, we first identify processes from a repository of available Web processes that match the query (cf. line 1 in Algorithm 1). Here, we assume that there is a matchmaker available that is capable to discover Web processes fulfilling given constraints (refer to Section II-C). Compositions of Web processes are investigated subsequently (see the for loop beginning at line 4). Therefore, for any binary disjunct decomposition of the set $\mathcal{O}$ of desired outputs, which is distinct from $\{\}$ and $\mathcal{O}$, the matchmaker identifies Web processes that fulfill the query with a reduced set of outputs. Divide and conquer allows us to recursively relax the query, which in turn decreases the complexity of the synthesis problem, because the number of desired outcomes is reduced while the remaining parameters are fixed.

1) Creating the Structure of the Controlling Process: The matchmaker always returns a set $\mathcal{P}$ of matching

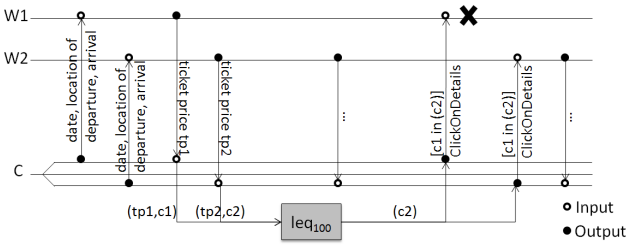

Figure 1. Synthesized solution template with a filter derived from a constraint.

Web processes. Each process $P \in \mathcal{P}$ runs in parallel in the created template. Then a controlling process C is synthesized such that each of the processes $P \in \mathcal{P}$ can be executed, i.e. inputs are provided and outputs are received by C. Therefore C splits into $|\mathcal{P}|$ threads; one thread C_P per process $P \in \mathcal{P}$. A thread C_P has roughly the complement interface of P. For example, there is always an input operation in C_P expecting the outputs from an output operation in P. Analogously, for each input operation $c[i_1, i_2, \ldots, i_m]$ in P there is an output operation $c < o_1, o_2, \ldots, o_m >$ in C_P providing the required inputs $i_1, i_2, \ldots, i_m$ at the time P is expecting the input on channel c. If P performs a local operation, C_P provides inputs and subsequently accepts outputs according to the signature of the local operation. In cases of deterministic choice, composition, and summation in P, the thread C_P is split again and the sub threads are analogously synthesized for the remaining sub processes.

2) Incorporating Atomic Web Services: If a matching process $P \in \mathcal{P}$, i.e. P provides the required outputs and requires inputs that are not supposed to be provided by the user manually (by comparing with $\mathcal{I}$), then the algorithm tries to provide a composition of atomic Web services that derives the missing inputs of P from the given information in $\mathcal{I}$. We assume, that if a user wishes some locally available information to be integrated automatically, there are corresponding Web services to access the information. Web services provide simple operations that can be used to transform the data; a composition of Web services can provide the required input for P. We will not describe further details of Web service composition, since many AI planning based techniques have been introduced and well-investigated [17], [18]. Our matchmaker therefore also need to consider temporal constraints on the processes used for the data transformation as we only consider those that provide outputs before another input is required that cannot be provided by C. For instance, a process requires airport codes as inputs instead of city names and an available service translates a city name to the airport code. Then the template should not introduce this service if the airport code is returned after the credit card is charged.

3) Example: Recalling Mary's travel booking scenario, we assume for example that there is no process that provides all required outputs. We focus on two component processes that provide flight arrangements. The controlling process operates them as shown

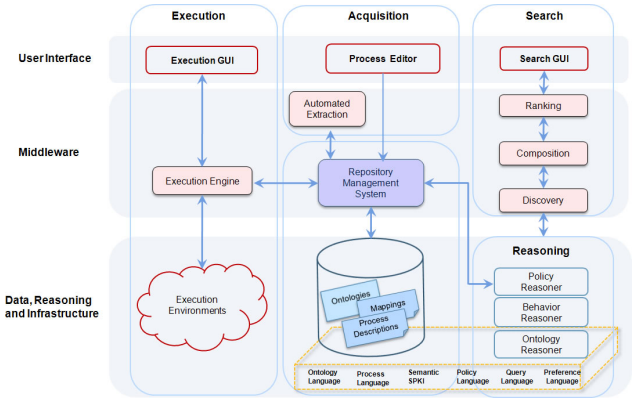

Figure 2. suprime Architecture

in Figure 1. Besides the synthesis of the controlling process as the complement of the individual Web processes, the controlling process should ensure that constraints and requirements of the query are taken into account. For a constraint $\gamma \in \Gamma$ that constraints the values of one required output $o \in \mathcal{O}$ a local process $l_\gamma(\{o_1, c_1\}, \{o_2, c_2\}, \ldots, \{o_n, c_n\})(c_1, c_2, \ldots, c_k)$ is added to a new thread of the controlling process C. After a process P_i returns o_i to the corresponding thread C_{P_i} of the controlling process and P_i's next process is an input operation, C_{P_i} forwards o_i and its communication channel identifier c_i to the local process l_γ that implements the process filter that is used for γ. The filter l_γ returns a set of communication channels $\{c_1, c_2, \ldots, c_k\} \subseteq \{c_1, c_2, \ldots, c_n\}$. In order to avoid synchronization issues, the channels are defined as shared variables within the scope of C. A channel $c_i \in \{c_1, c_2, \ldots, c_k\}$ is in the result set if the corresponding process P_i is continued. We say that a process P is continued if the thread C_P provides inputs to the next input operation in P after P returned the parameter required for the filter. $C_{P'}$ does not provide any further inputs to P' if P' is not continued. As shown in Figure 1, after the price information of offered flights were received by the threads of the controlling process, a filter leq_{100} synchronizes both threads by receiving price information fp from each thread in C. Mary specified the requirement $\leq (\mathsf{fp}, 100) \in \Gamma$, which means that processes with prices lower than 100 may continue execution at runtime.

4) Ranking: Handling preferences Π^o on outputs, like the ticket prices of offered flight, are similarly handled within the synthesis of a controlling process. A synchronizing activity is introduced after the Web processes provided the parameters and computes the ranking. Then, a ranked list of the results is generated with the help of the ranking technique introduced in Section II-D and displayed in the browser, which lets the user select one or multiple outputs, e.g. flight offers, that can be further considered in the remaining process, e.g. hotel and rental car reservation.

IV. Implementation

In this section, we give overviews of the implementation of the suprime components relevant for this paper and refer to the suprime Web site [2] for more technical details. Figure 2 shows the main components of the suprime framework.

The languages for describing processes, offers, queries and preferences build the basis for the intelligent techniques like acquisition, search, composition, ranking and execution. Our process description language *suprimePDL* has been briefly introduced in Section II-A. The query language for specifying constraints on process properties, especially temporal constraints, has been presented in [19]. The Fuzzy If-Then rules based preference specification language has been introduced in [16]. For each language, we have developed a Java API as well as a graphical notation.

The Repository component stores process descriptions, ontologies and ontology mappings persistently. The descriptions can be managed with the help of the methods for adding, removing and updating the descriptions. E.g. the automatic acquisition module that generates the semantic process descriptions as introduced in Section II-B uses these methods to manage the process descriptions persistently.

The Search component has direct access to the repository and searches for process descriptions within the repository that fulfill a query as introduced in Section II-C and presented in [19]. Roughly, the search is based on a tableaux based model checking algorithm that checks whether a process expression is a model of a temporal logic formula. In order to achieve efficiency, indexing techniques based on the simulation relationships among the process expression have been incorporated, which can be computed independent of a query, and therefore offline.

While the Search component filters the set of processes to only those that fulfill certain criteria, the Ranking component sorts the processes according to a preference structure. In our system, the preferences are defined as Fuzzy rules as introduced in Section II-D. We have implemented the FITA (First Inferencing Then Aggregation) strategy [20] in Java that computes a score between 0 and 1 for a given set of values for NFPs and a Fuzzy rule base.

The query formalism proposed in Section III for specifying end user requirements is roughly a union of the Fuzzy rules based preference specification formalism [16] and a subset (without temporal constraints) of the query language presented in [19]. Therefore, the Java APIs for the query language and the preference language are used to process the user's synthesis requirements programmatically. A user interface allows graphical modeling of user's requirements on a solution. The synthesis algorithms presented in Section III, implemented as part of the Composition component, generates a set of solution

[2] http://suprime.aifb.uni-karlsruhe.de

templates. For doing so, it often utilizes the search for appropriate processes in the repository.

The browser based front end is based on the open source Oryx process editor [3] and allows end users to model, search, compose Web processes graphically as well as execute them in the Web browser. We have extended the Oryx editor to support our languages by the so called Stencil Sets. In particular, the process editor allows users to refine the automatically obtained *suprimePDL* descriptions of the Web processes. The search GUI allows users to model a query in the above mentioned query language graphically. The discovery component returns the set of process descriptions that fulfill the query and the ranking GUI allows users to define Fuzzy sets and model their preferences as Fuzzy rules. When a user has modeled the requirements on a solution template, he can press the "synthesize" button, upon which the synthesis algorithm is invoked. The set of solution templates (*suprimePDL* process expressions) is sent to the ranking component together with the preferences, which return a sorted list of solution templates. The list is presented to the user at the GUI, where he can view the details of each solution template, refine the solution templates and store useful ones in the repository.

In our browser based implementation of the execution of a complex process, a browser tab corresponds to a thread. That is, for each thread a new browser tab is opened, in which the thread can receive inputs and provide outputs. When the thread terminates, the corresponding browser tab is closed. When an input is required from the user by a controlling process, an HTML form is created from the set of input variable and displayed in the corresponding tab. Similarly, when a controlling process produces an output to the user, the output values are displayed in the corresponding tab. If there is a data flow specified from an output activity of a controlling process to an input activity of a Web process, the values are entered in the corresponding form and the form is submitted automatically. Note that, in this way the names of the browser tabs act as communication channels for various input and output activities. When a non-deterministic choice is executed, a Web page is generated with a list of links and forms depending on whether an alternative is a process invocation or an input activity. A user can then either click on a link or submit a form. In case of a link selection the URL of the tab is changed to the URL of the new link, whereas in case of a form the tab behaves as described above.

V. Related Work

Annotation of Web pages has been of interest for quite some time now [2]. However, the main goal of the annotation approaches was to annotate the data on Web sites with ontologies to achieve better interoperability. We base our work on an approach that can capture not only dynamic Web pages, but more interestingly also the dynamics of the flow of Web pages. The semantic descriptions of the data on Web pages goes adjacent to the semantic description of the behavior of the Web sites. Our process description language *suprimePDL* is more appropriate than e.g. OWL-S[3] due to (1) its clear formal semantics and Turing complete expressivity and (2) its support for mobility which makes it possible to send links as data objects which is inherent in Web processes.

From the research community, perhaps the METEOR-S project first used the term Web processes, even though with a different meaning than we used it in this paper [21]. In [21] and other many other related METEOR-S research works, the focus is on constructing workflows by combining atomic Web services. In this paper, our focus is on viewing Web sites as processes and combining them to (more complex) processes. Mashup tools, e.g. Yahoo! Pipes[4] allow users to combine data from various Web pages and present the aggregated view on the data on a new Web page. Thus, mashup tools are data flow driven. Furthermore, they mostly rely on RESTful Web services for obtaining access to the data. Our main focus in this paper is not to create a new Web pages with aggregated information collected from various Web pages, but rather to provide users of the Web sites with techniques for composition and execution of Web sites in order to relieve them from manual coordination of the Web sites they often use for a task at hand.

iMacros[5] is a commercial browser plugin that allows users to record a navigation behavior as a macro and execute such macros at some later stage. However, the synthesis of the macros is fully manual and a macro can use only those Web processes that are known to the end user while recording it. Our automatic synthesis technique allows users to compute generic solutions based on user's requirement automatically. While doing so, all Web process descriptions available in the repository can be used. Furthermore, the formal nature of our process language makes it possible to employ automatic procedures for reasoning about the properties of synthesized solution templates. In the recent years, many automatic composition techniques, e.g. [5], have been proposed. The major difference between them and our synthesis approach is that we aim at gluing together processes to a more complex process, whereas automatic composition techniques aim at composing atomic Web services to workflows.

The execution with iMacros is rather syntactic, by which we mean that it does not support interoperability of data among different sources by considering their semantics. In our approach, we rely on semantic descriptions of the data with ontologies with a standard language OWL in order to achieve the semantic interoperability despite differences in the terminologies used at different Web sites. Furthermore, our execution engine generates HTML forms for receiving user inputs that can be used across all the involved Web processes.

[3]http://bpt.hpi.uni-potsdam.de/Oryx/WebHome

[4]http://pipes.yahoo.com/pipes/

[5]http://www.iopus.com/iMacros/

VI. Conclusion and Outlook

The work presented in this paper was motivated by mainly two observations (1) Most of the interesting processes in the Web are targeted at human users and (2) human users have to coordinate various Web processes manually. We have argued that most of such coordination e.g. entering the same or logical dependent data in many different forms, can be automatized. In this paper, we have presented an approach that helps the users to automatize the coordination, which is especially beneficial in case of recurring tasks. We first presented the overviews of the techniques that are used in the main part of the paper. Our main contribution lies in the interplay of many techniques to obtain a useful application in the wider sense, as well as in the automatic technique for supporting users in the synthesis of solution template in the deeper sense. While composition is an intermediate step, the ultimate goal of the user is to achieve efficiency in day to day work by executing the solution templates. We addressed this issue by presenting a Web browser based execution environment that automates navigation of Web processes while still allowing manual interaction in case where input from human is required or desired by the user. Even though all the necessary components of our approach are implemented, they still need to be packed in a browser plugin. We plan to implement the browser plugin and make it publicly available soon. Furthermore, scalability of the system is an important issue, that we need to investigate in the near future, since we consider Web processes which are huge in number.

References

[1] N. Steinmetz, H. Lausen, and M. Brunner, "Web Service Search on Large Scale," in *ICSOC/ServiceWave*, ser. Lecture Notes in Computer Science, L. Baresi, C.-H. Chi, and J. Suzuki, Eds., vol. 5900, 2009, pp. 437–444.

[2] S. Handschuh and S. Staab, Eds., *Annotation for the Semantic Web*. IOS, 2003.

[3] K. P. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, "Automated discovery, interaction and composition of Semantic Web services," *Web Semantics*, vol. 1, no. 1, pp. 27–46, 2003.

[4] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Pollers, C. Feier, C. Bussler, and D. Fensel, "Web Service Modeling Ontology," *Applied Ontology*, vol. 1, pp. 77–106, 2005.

[5] J. A. Baier, F. Bacchus, and S. A. McIlraith, "A Heuristic Search Approach to Planning with Temporally Extended Preferences," *Artificial Intelligence*, vol. 173, no. 5-6, pp. 593–618, 2009.

[6] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. P. Sycara, "The DAML-S Virtual Machine," in *International Semantic Web Conference*, 2003, pp. 290–305.

[7] T. Vitvar, A. Mocan, M. Kerrigan, M. Zaremba, M. Zaremba, M. Moran, E. Cimpian, T. Haselwanter, and D. Fensel, "Semantically-enabled Service Oriented Architecture: Concepts, Technology and Application," *In Journal of Service Oriented Computing and Applications*, 2007.

[8] S. Agarwal, "Formal Description of Web Services for Expressive Matchmaking," Ph.D. dissertation, University of Karlsruhe, Universit{ä}t Karlsruhe (TH), Institut AIFB, D-76128 Karlsruhe, 2007.

[9] S. Agarwal, S. Rudolph, and A. Abecker, "Semantic Description of Distributed Business Processes," in *AAAI Spring Symposium - AI Meets Business Rules and Process Management*, Stanford, USA, 2008.

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

[11] M. Ehrig, S. Staab, and Y. Sure, "Bootstrapping Ontology Alignment Methods with APFEL," in *Proceedings of the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005.*, ser. LNCS, vol. 3729, November 2005, pp. 186–200.

[12] S. Agarwal, "Semi-Automatic Acquisition of Semantic Descriptions of Web Sites," in *Proceedings of The Third International Conference on Advances in Semantic Processing (SEMAPRO)*. Malta: IEEE, 2009.

[13] J. Hoxha and S. Agarwal, "Semi-automatic Mining of Semantic Descriptions of Processes in the Web," in *IEEE/WIC/ACM International Conference on Web Intelligence*. Toronto, Canada: IEEE, Aug-Sep 2010.

[14] 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*, vol. 7, pp. 11–20, Januar 2009.

[15] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-3, pp. 28–44, 1973.

[16] I. Toma, N. Steinmetz, H. Lausen, S. Agarwal, and M. Junghans, "First Service Ranking Prototype - SOA4All Deliverable 5.4.1," 2010.

[17] F. Lécué, "Optimizing QoS-Aware Semantic Web Service Composition," in *ISWC '09: Proceedings of the 8th International Semantic Web Conference*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 375–391.

[18] S. Sohrabi and S. A. McIlraith, "Optimizing Web Service Composition while Enforcing Regulations," in *Proceedings of the 8th International Semantic Web Conference (ISWC09)*, 2009, pp. 601–617.

[19] S. Agarwal, "A Goal Specification Language for Automated Discovery and Composition of Web Services," in *International Conference on Web Intelligence (WI '07)*, T. Y. Lin, L. Haas, J. Kacprzyk, R. Motwani, A. Broder, and H. Po, Eds., Silicon Valley, California, USA, November 2007, pp. 528–534.

[20] K.-H. Temme and H. Thiele, "On the correctness of the principles of FATI and FITA and their equivalence," in *IFSA 95 - Sixth International Fuzzy Systems Association World Congress*, vol. II, 1995, pp. 475–478.

[21] J. Cardoso and A. P. Sheth, "Semantic E-Workflow Composition," *J. Intell. Inf. Syst.*, vol. 21, no. 3, pp. 191–225, 2003.