iBookmarks: Synthesis and Execution of Solution Templates for efficient Usage of recurring Web-Process Combinations

Sudhir Agarwal
Karlsruhe Institute of Technology (KIT),
Institute of Applied Informatics and Formal Description Methods (AIFB),
Karlsruhe Service Research Institute (KSRI),
Englerstr. 11, Karlsruhe, Germany.
Email: sudhir.agarwal@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 sites. However, currently the composition of Web sites, coordination of the execution of such Web sites compositions is done completely manually. In this paper, we present an approach that allows users to automatically compose generic solutions by combining appropriate Web sites and invoke the generic solutions with appropriate parameters whenever required, thus relieving them from a lot of manual coordination effort. We show how Web sites and their compositions can be formalized as processes, how the formal descriptions of Web sites can be automatically composed to obtain generic solutions and how such generic solution can be 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 and Web sites.

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.

a) 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-0a), 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

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

best of our knowledge, the composite service description techniques such as OWL-S Process Model have not been applied for describing dynamics of Web sites. Unfortunately, the plethora of automatic Web service composition approaches focus only on atomic Web services, e.g. [5], [8], [9], [27] are not applicable for composing processes. As a consequence there is also a lack of a semantic execution engine for executing and coordinating Web processes.

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*, which are complex decentralized workflows with Web processes as its components, and remember the solution templates as "intelligent" bookmarks in the Web browser. Furthermore, a user will be able to select the bookmark appropriate for a concrete instantiation of a problem, which triggers the execution of the complex workflow underlying the solution template. In order to be able to do composition and execution of solution templates, descriptions of Web processes must be available. In Section II-A and Section II-B we give overviews of our Web process description formalism as well as our semi-automatic technique for obtaining descriptions of the processes implicit in the flow of Web pages. Furthermore, composition of solution templates relies on retrieval of appropriate Web process descriptions from a repository of Web process descriptions, which we introduce in Section II-C. 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.

Name	Syntax	Semantics
Null	$\mathbf{0}$	does nothing; used as termination symbol
Input	$\mathsf{c}[\mathbf{x}].P$	takes inputs at port c, binds them to variables $\mathbf{x}$ and then behaves like P
Output	$\mathsf{c}\langle\mathbf{y}\rangle.P$	outputs the values $\mathbf{y}$ at port c and then behaves like P
Local	$\Delta.P$	performs the list of changes Δ and then behaves like P
Conditional	$\omega?P$	behaves like P if condition ω can be evaluated to true, otherwise like $\mathbf{0}$
Composition	$\prod_{1\leq i\leq n} P_i$	parallel composition of n process components P_i
Choice	$\sum_{1\leq i\leq n} P_i$	behaves like exactly one of the n alternative processes P_i
Agent Invocation	$@A\{\mathbf{y}\}$	invocation of an agent identifier A with arguments values $\mathbf{y}$. An agent identifier A with arguments $\mathbf{x}$ is defined with a process expression in which $\mathbf{x}$ are the only names that may occur freely.

TABLE I
π-CALCULUS SYNTAX AND SEMANTICS OF BEHAVIOR DESCRIPTIONS

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 II
CORRESPONDENCE BETWEEN PAGE CONTENT AND ONTOLOGY

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 [10], [11].

The behavior ϕ_D of a Web process is described with the π-calculus process algebra [12] in combination with a semantic description of static and dynamic process resources in the domain ontology O_D expressed in $\mathcal{SHIQ}(\mathbf{D})$ description logic expressions. E.g., input parameters $\mathbf{x}$ and the communication channel c are resources and further described in O_D (cf. Table I). Benefits of this combined approach, details on the description formalism, and its formal semantics are introduced in [13]. Here, we give an overview about the syntax and its semantics in Table I. The semantics of π-calculus process expressions is defined on a labeled transition system (LTS) the knowledge of the service in that stage of the execution and is described by an ontology. The ABox of the ontology is subject to change during state transitions and the TBox is assumed to be invariant during execution.

B. View of Web Sites as *suprimePDL* Processes

Table II 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

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 III
MAPPING BETWEEN WEB ARTIFACTS AND ELEMENTS OF THE FORMALISM

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. [14]). 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 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. [15] in such a large pool of ontologies are necessary.

A set of mappings, illustrated in Table III, 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:

$$\mathsf{y}\langle \mathsf{o}_1, \ldots, \mathsf{o}_l \rangle.\mathbf{0} \parallel \{U_1 + \ldots + U_l + F_1.N_1 + \ldots + F_n.N_n\},$$

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* [16], [17].

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 processes that can be glued together to a given process. In [18], 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 [19], [20] temporal logic and $\mathcal{SHIQ}(\mathbf{D})$ description logic. Desired exchanged messages, i.e., input and output parameters, are constrained in a $\mathcal{SHIQ}(\mathbf{D})$ domain ontology $\mathcal{O}_{\mathcal{R}}$, which allows us to express assumptions on the types of the messages. Then, the desired behavior is modeled by expressions using the μ-calculus. The modal μ-calculus, an extension of the modal logic, is used to query for modal and temporal properties of processes expressions. It has a simple syntax, an easily given semantics, and the fixpoint operators provide immense power [19], [20].

Definition 1: Basic μ-calculus Syntax

$$\Psi := \Psi \wedge \Psi \mid \neg\Psi \mid \mu X.\Psi(X) \mid \langle a \rangle \Psi \mid P \mid \top \mid \bot$$

Conjunction and negation allow to compose inclusions and exclusions of desired process fragments. The terminals of the expression are the propositions P, existence of an action a (e.g., occurrence of input and output action), as well as $\top$ and $\bot$, which match all or no processes, resp. The minimal fixpoint operator allow to specify formulas recursively. Note that disjunction, universal quantifier for actions as well as maximal fixpoint operator can be built using the above basic constructs. However, since the specification of temporal constraints using fixpoint operators is very technical and likely not to be handled by end users, we extend the constraint specification language by some commonly used patterns such as $\mathbf{until}$, $\mathbf{eventually}$, and $\mathbf{always}$ as predefined patterns that can be expressed with the fixpoint operators ψ_1 $\mathbf{until}$ ψ_2 means that the process will reach some state in which ψ_2 holds and ψ_1 holds until this state is reached. A process that must reach a state in which ψ holds is expressed by $\mathbf{eventually}$ ψ. $\mathbf{always}$ ψ means that ψ holds on every path, whereas further restriction to the path may be applied. We refer to [19], [20] for further details on the syntax and semantics of the μ-calculus.

For the 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 refer to [18]. 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.

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 [17]. 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. Structure of the Controlling Processes

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.

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 $(\mathcal{I}, \mathcal{O}, \Gamma)$ that contains a set $\mathcal{I}$ of inputs, a set $\mathcal{O}$ of outputs, a description Γ of constraints on the inputs and outputs including their types and how they relate with each other.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. In our example, Mary wants to provide inputs

$$\begin{aligned}\mathcal{I} \quad = \quad & \{\mathsf{flightArrivalTime}, \mathsf{flightStart}, \mathsf{flightEnd}, \\ & \mathsf{carPickupTime}, \mathsf{carPickup}, \mathsf{creditCard}, \mathsf{user}\}\end{aligned}$$

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 describe this as

$$\mathcal{O} = \{\mathsf{ft}, \mathsf{ftEndTime}, \mathsf{fp}, \mathsf{car}, \mathsf{cp}\}.$$

Γ denotes a logical expression that describes the types of 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. In our example,

$$\begin{aligned}\Gamma = \{ & \mathsf{Time}(\mathsf{flightArrivalTime}), \mathsf{Airport}(\mathsf{flightStart}), \mathsf{Airport}(\mathsf{flightEnd}), \\ & \mathsf{Time}(\mathsf{carPickupTime}), \mathsf{City}(\mathsf{carPickup}), \mathsf{CreditCard}(\mathsf{creditCard}), \\ & \mathsf{UserProfile}(\mathsf{user}), \mathsf{FlightTicket}(\mathsf{ft}), \mathsf{Time}(\mathsf{ftEndTime}), \\ & \mathsf{RentalContract}(\mathsf{car}), \mathsf{equiv}(\mathsf{flightEnd}, \mathsf{ftEnd}), \mathsf{Price}(\mathsf{cp}), \\ & \mathsf{before}(\mathsf{ftEndTime}, \mathsf{carPickupTime}), \mathsf{Price}(\mathsf{fp}), \leq (\mathsf{fp}, 100)\}\end{aligned}$$

is a set of constraints that are interpreted as a conjunctive query. Each constraint can be a binary or unary predicate. $\mathsf{equiv}(\mathsf{flightEnd}, \mathsf{ftEnd})$ states that the desired destination equals the destination airport of the flight ticket. The constraint $\mathsf{before}(\mathsf{ftEndTime}, \mathsf{carPickupTime})$ 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.

Given desired properties $(\mathcal{I}, \mathcal{O}, \Gamma)$, 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 the *match* method provides the functionality of the matchmaker to discover Web processes fulfilling given constraints (refer to Section II-C). Compositions of Web processes are computed with a Plan Space Planning algorithm [21]. The algorithm takes the properties of the desired solution templates and a set of already created solution templates as input. It selects randomly one constraint γ from the set of desired constraints and tries extends the already created partial solution templates such that the new partial solution templates fulfill the constraint. The constraint is then removed in order to ensure that it is not considered again unless it is added as part of the preconditions of the newly added process fragments. The algorithm invoked itself with the new set of solution templates and the new set of constraints until there are no unsatisfied constraints left. In the beginning, the algorithm is called the complete set of constraints as specified by the user and an empty set of solution

Algorithm 1 composeSolutionTemplate

Require: Desired properties $(\mathcal{I}, \mathcal{O}, \Gamma)$ and a set of solution templates $\mathcal{S}$

```
if Γ = ∅ then
    Stop!
take one γ ∈ Γ
let S' ⊆ S denote those solution templates that do not satisfy γ
if S' ≠ ∅ then
    P_match ← match(γ)
    for all P ∈ P_match do
        for all S ∈ S' do
            copy S to S'
            let C denotes the controlling process of S'
            add P ∥ C
            adjust C
            add φ_P, the preconditions of P, to Γ
        remove S from S'
    if S' has changed then
        remove γ from Γ
        composeSolutionTemplate(S', (I, O, Γ))
    else
        Failure!
else
    remove γ from Γ
    composeSolutionTemplate(S, (I, O, Γ))
```

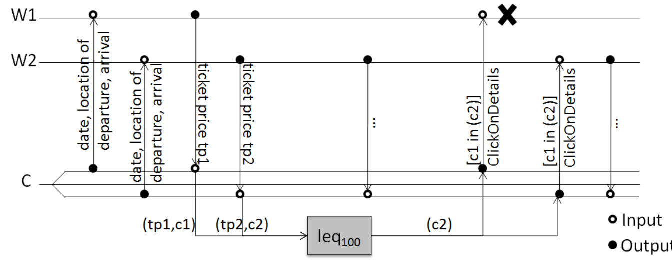

Fig. 1. Synthesized solution template with a filter derived from a constraint.

templates. Once all the solution templates are synthesized such that each of the synthesized solution template fulfills the constraints specified in Γ, the final set of solution templates is obtained by selecting only those that have desired inputs and outputs as specified in $\mathcal{I}$ and $\mathcal{O}$. This selection is performed with the help of the same model checking algorithm *match* that is used for searching processes.

1) Creating and Adjusting the Structure of the Controlling Process: The synthesis algorithm returns a set of solution templates. The component processes $\mathcal{P}$ of a solution template run 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 $P \in \mathcal{P}$. The interface (choreography) of a thread C_P is complement of that of P. That is, C_P has a corresponding input operation for each output operation in P, and a corresponding output operation providing the required inputs for each input operation of P in the right order and at right channel to ensure the data flow. 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 [22], [23]. 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 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

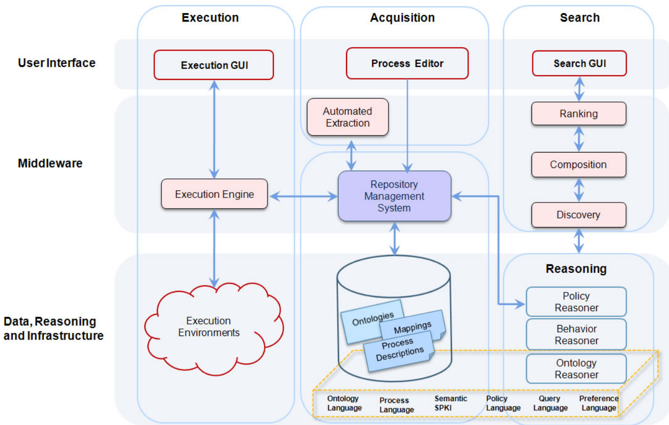

Fig. 2. suprime Architecture

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.

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 [24]. The Fuzzy If-Then rules based preference specification language has been introduced in [25]. 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 [24]. 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 off-line.

The query formalism proposed in Section III for specifying end user requirements is roughly a union of the Fuzzy rules based preference specification formalism [25] and a subset (without temporal constraints) of the query language presented in [24]. 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 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

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

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

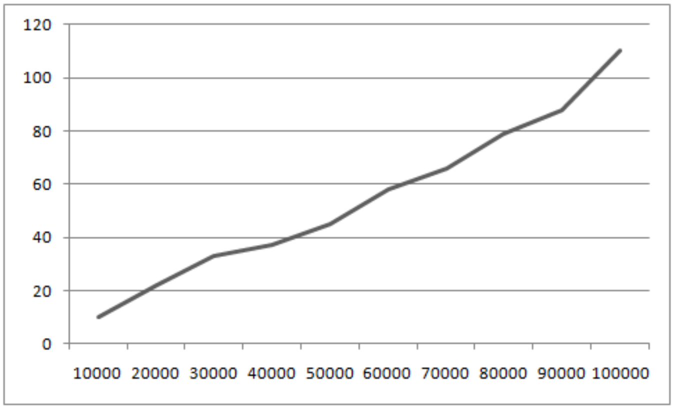

Fig. 3. Performance Evaluation of composition (Number of total Web process descriptions in the repository vs. mean time in seconds for computing the compositions for the example query.

of the tab is changed to the URL of the new link, whereas in case of a form the tab behaves as described above.

Figure 3 shows the results of the performance evaluation of the composition algorithm. We a collection of 100,000 Web process descriptions created by our semi-automatic acquisition technique presented in Section II. For the purpose of evaluation we have created increasingly larger chunks varying from 10,000 to 100,000 by randomly selecting the descriptions from the original set of 100,000 descriptions. Note, that the performance for composition to satisfy our example query which was from the traveling domain depends primarily on the number of processes from the traveling and not on the total number of processes. Still the later plays an important role since the ontology reasoners and the temporal logic reasoner have to first load the whole repository in memory.

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 [26]. In [26] 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 Web Service Composition (WSC) techniques, e.g. [5], have been proposed. A popular approach to WSC is to characterize it as an Artificial Intelligence (AI) planning task and to solve it as such (e.g., [8], [9], [27]). 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. Several approaches have been proposed to deal with composition of complex processes, e.g. [28], [29] and a comprehensive comparison is hard since they address different flavors of the composition problem. One of the main distinction to be done is between *centralized* (or mediated, orchestrated) composition methods and *distributed* (or peer-to-peer) methods. The difference lies in the automated composition result: the former aim at synthesizing a new service (*mediator*) that orchestrates the component services by properly exchanging messages, while in the latter the execution of the composition is distributed among all the component services.

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.

[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. In future, we will continue to work on the scalability issues of search and composition of Web processes.

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] D. V. McDermott, "Estimated-regression planning for interactions with web services," in *AIPS*, M. Ghallab, J. Hertzberg, and P. Traverso, Eds. AAAI, 2002, pp. 204–211.

[9] S. A. McIlraith and T. C. Son, "Adapting golog for composition of semantic web services," in *KR*, D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, Eds. Morgan Kaufmann, 2002, pp. 482–496.

[10] S. Agarwal, "Formal Description of Web Services for Expressive Matchmaking," Ph.D. dissertation, University of Karlsruhe, Universität Karlsruhe (TH), Institut AIFB, D-76128 Karlsruhe, 2007.

[11] S. Agarwal, S. Rudolph, and A. Abecker, "Semantic Description of Distributed Business Processes," in *AAAI Spring Symposium - AI Meets Business Rules and Process Management*, K. Hinkelmann, A. Abecker, H. Boley, J. Hall, M. Hepp, A. Sheth, and B. Thönssen, Eds., Stanford, USA, 2008.

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

[13] S. Agarwal, S. Rudolph, and A. Abecker, "Semantic Description of Distributed Business Processes ," in *In Knut Hinkelmann, Andreas Abecker, Harold Boley, John Hall, Martin Hepp, Amit Sheth, Barbara Thönssen, AAAI Spring Symposium - AI Meets Business Rules and Process Management*, 2008.

[14] 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.

[15] 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, E. Motta, Y. Gil, M. A. Musen, and V. R. Benjamins, Eds., November 2005, pp. 186–200.

[16] 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.

[17] 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.

[18] 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.

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

[20] J. Bradfield and C. Stirling, "Modal Logics and mu-Calculi: An Introduction," in *Handbook of Process Algebra*, J. A. Bergstra, A. Ponse, and S. A. Smolka, Eds. New York, NY, USA: Elsevier Science Inc., 2001, pp. 293–330.

[21] D. A. McAllester and D. Rosenblitt, "Systematic nonlinear planning," in *AAAI*, 1991, pp. 634–639.

[22] 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.

[23] 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.

[24] 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.

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

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

[27] P. Bertoli, M. Pistore, and P. Traverso, "Automated composition of web services via planning in asynchronous domains," *Artif. Intell.*, vol. 174, no. 3-4, pp. 316–361, 2010.

[28] D. Berardi, D. Calvanese, G. D. Giacomo, and M. Mecella, "Composition of services with nondeterministic observable behavior," in *ICSOC*, ser. Lecture Notes in Computer Science, B. Benatallah, F. Casati, and P. Traverso, Eds., vol. 3826. Springer, 2005, pp. 520–526.

[29] S. A. McIlraith and R. Fadel, "Planning with complex actions," in *NMR*, S. Benferhat and E. Giunchiglia, Eds., 2002, pp. 356–364.