Expressiveness of Automatic Semantic Web Service Composition Approaches: A Survey based on Workflow Patterns

Evando Carlos Pessini^{1 2 3}

Abstract: Este artigo apresenta uma avaliação comparativa das principais técnicas empregadas para a geração automática de composições de serviços web semânticos. A avaliação usa como base os diferentes padrões de workflow suportados por cada técnica nas soluções (composições) geradas. A análise considera abordagens que geram composições de forma automática, ou seja, técnicas de composição que não requerem intervenção do usuário no processo de composição ou que demandam entradas complexas (como por exemplo, templates ou outras formas de modelos de composição abstratos). A contribuição deste artigo se centra na discussão dos pontos fortes e das deficiências das abordagens apresentadas na literatura para a geração automática de composições de serviços web semânticos a partir da perspectiva dos padrões de workflow.

Abstract: This survey evaluates a number of techniques for the automatic composition of Semantic Web Services. The evaluation takes into account the different workflow patterns that can be expressed by the solutions proposed by each individual technique. Only fully automatic approaches are considered here, i.e., composition techniques that do not demand user interference in the composition process nor rely on complex input (such as templates or other forms of abstract composition models). The main contribution of this paper is to discuss the strengths and weaknesses of the approaches presented in the literature for the automatic generation of semantic web services composition from the workflow patterns perspective.

1 Introduction

Service composition has received much attention in the last decade. This is due to its potential to build complex web service-based applications from elementary component services, thus, causing a considerable momentum to Service Oriented Computing [19]. During this period, a large number of research projects have treated with the service composition

¹Núcleo de Ciência da Computação, UTFPR/Medianeira - Paraná - Brasil
{pessini@utfpr.edu.br}

²Departamento de Matemática e Informática Aplicada (DIMAp), UFRN - Natal - Brasil

³Instituto de Computación (InCo), UdelAR - Montevideo - Uruguai

problem, either as main focus or as secondary one. Examples of these research projects include SOA4All⁴, DIP⁵, SUPER⁶ and SHAPE⁷. These projects as well other minor research projects employ models and techniques from many areas of computer science for formalizing the composition problem and generating the service compositions, giving rise to a puzzling scenario. These models and techniques exploit the knowledge produced in research areas such as, AI-planning, theorem proving, logical satisfiability, graph theory, process algebra, to name a few.

Owing to the increased interest in the subject, several papers were published concerned with establishing requirements and characteristics to evaluate and to classify the web service composition methods. Former surveys [58, 55] show that most composition methods employ AI-based techniques. The work presented in [47] aims at providing foundations to analyse the required functionality of service composition methods, in order to automatically compose real-world services. That work also establishes requirements for the *composition elements*, *control flow*, *data flow* and *data model* of compositions. The work presented in [21] treats the dynamic aspect of the composition approaches, aiming to define criteria to identify the levels of dynamicity and automation in service compositions.

The authors in [8] present a comprehensive and comparative analysis of the most representative efforts in automated web service composition, considering requirements such as *automation level*, *dynamicity*, *semantic capabilities*, *QoS awareness*, *non-determinism*, *partial observability*, *scalability*, *correctness*, *domain independence* and *adaptivity*. Also [7] provides an overview of several existing approaches dealing with the web service composition problem, putting special emphasis on issues such as the design of the control-/data-flow, functional extension, QoS, optimal compositions and performance evaluation. The work presented in [12] is more closely related to the goals of this survey, in the sense that it presents an analysis of several automatic service composition approaches. However, that work treats only methods based on AI-planning techniques for service composition. The present survey complements the ones cited above, by dealing with a number of other techniques, including mathematical-based and model-based techniques. Furthermore, some other works that do not fit in the previously mentioned categories are grouped in a category called *Others*.

The contributions of this paper are twofold. Firstly, it provides a tabular view illustrating the workflow patterns supported by each approach. It focuses on the perspective of workflow patterns to evaluate the expressiveness of the different approaches. Secondly, it discusses the strengths and weaknesses of the techniques and underlying formalisms employed by approaches to automatically generate web services compositions.

⁴<http://www.soa4all.eu/>

⁵<http://dip.semanticweb.org/>

⁶<http://www.ip-super.org/>

⁷<http://www.shape-project.eu/>

This survey is mainly characterized by presenting a broader literature review than other related surveys, encompassing approaches that employ very different strategies and techniques for generating service compositions. Its other unique feature is to evaluate the ability of the approaches for generating compositions containing the well-established workflow patterns [68]. The results presented here are important for establishing a synthetic panorama of the approaches' potentialities in addition to providing an updated literature review on this research subject which has been extensively studied in the last decade.

The paper is structured as follows: Section 2 provides a description of the basic topics underlying the semantic web service composition problem, as well the definition and exemplification of the composition problem itself. Section 3 describes the state-of-art of semantic web service composition methods with focus on the supported workflow patterns. The results of analysing those approaches are depicted in table 4, which presents a synthetic panorama of each work, versus their supported workflow patterns. Then, section 4 discusses the strengths and weakness of the approaches. Section 5 concludes this work.

2 Preliminaries

This section is intended to present the central concepts of the semantic web services, as well a small example aiming to illustrate how they are used for solving the automatic service composition problem.

Semantic Web Service Semantic web services were designed to overcome some limitations of the initially proposed technologies for web services. These limitations concern to the automation of activities involving its usage, for instance, discovery and composition. The major limitation is related to the language employed for describing the web services, namely WSDL [14], which allows service interface to be only described syntactically. Thus, different behaviorally services having the same interface (WSDL description) cannot be automatically and accurately differentiated harming the automation of such activities.

Aiming to increase the automation level of web services usage, a plenty of technologies for semantic web services were development in the last decade. Particularly, service ontologies such as OWL-S [45] and WSMO [17], were developed to allow web services have well-defined functional semantic, making possible to automate activities as mentioned above. In general, semantic web services are defined by a 4-tuple $S = \{I, O, Pre, E\}$, where I and O stand for two sets of parameters whereas Pre and E stand for conditions over the world state required to hold before the service execution (preconditions) and after the service execution (effects). Each parameter in the sets I and O is associated to an ontological concept

$$\begin{aligned}
 \textit{CreditCard} &\sqsubseteq \textit{PaymentMethod} \\
 \textit{Authorization} &\sqsubseteq \textit{PaymentMethod} \\
 \textit{ChargedCard} &\equiv \textit{CreditCard} \sqcap \exists \textit{hasCharge.Amount} \\
 \textit{Price} &\sqsubseteq \textit{Amount}
 \end{aligned}$$

Figure 1. Domain ontology

defined in some domain ontology⁸. Conditions are expressed by formulae in some logical language. Description logic [3] has widely been used as underlying formalism for representing and reasoning about domain knowledge.

In order to illustrate how semantic web service is defined, the description of a service charging a given credit card (such as PayPal⁹) is presented bellow . For sake of readability and neutrality, no service ontology was used for describing the semantic web services. Instead, their elements (inputs, outputs, preconditions and effects) are described directly through of formulae in description logic. Concepts and properties used to annotate the service elements are defined in the following simplified ontology.

Table 1. PayPal Service

Element#1	Name#2	Type#3
Inputs	card	CreditCard
	amount	Amount
Outputs	auth	Authorization
Preconditions		
Effects	charged	$\textit{hasCharge}(\textit{card}, \textit{amount}), \textit{hasValidCode}(\textit{auth}, \textit{someCode})$

Both services and ontology will be used for illustrating the semantic web service composition problem at the end of this section.

Workflow Patterns Process-based information systems have increasingly been adopted by enterprises as a mechanism to align their businesses with the available technological resources. This alignment improves enterprises' capabilities for adapting faster and cheaper their technological resources in response to changes in their business policies. Processes play

⁸An ontology presents a shared vocabulary of terms employed in conceptualization of the given domain [25]. Other definitions of ontology can be found in [26, 27]

⁹PayPal is a registered trademark of PayPal, Inc.

Table 2. Online Shop Service

Element#1	Name#2	Type#3
Inputs	product	Product
	payment	PaymentMethod
Outputs	invoice	Invoice
	delivery	Delivery
Preconditions	paymentValid	$\exists hasValidCode.\top(payment)$
Effect	scheduledDelivery	$hasDeliveryInProgress(invoice, delivery)$

a central role in this context representing the business logic of the enterprise by interplaying between its tasks, roles and resources. This special kind of information systems can be seen from some different perspectives according to the observer's interest. Two of the more meaningful perspectives are the *control flow* and *data flow*. Whereas control flow perspective treats with the control flow dependencies between tasks, data flow perspective treats how data are handled/exchanged between the tasks.

Aiming to standardize the evaluation of workflow management systems, a collection of patterns was proposed in [68] concerning mainly to the control and data flow perspectives. Only patterns related to the control flow perspective are considered here because the objectives of this survey. The following is a brief description of the most common control flow patterns used in the definition of business processes (workflows).

Sequence pattern represents the sequential execution of tasks, where a task is only executed after the execution termination of the precedent task. This is a common control flow pattern presents in imperative programming languages. **AND-split -join** pattern represents the parallel execution of services where all branches are simultaneously executed. Its counterpart JOIN represents the synchronization constructor, which requires the execution termination of all parallel branches (synchronization) in order executing the next task. **XOR-split -join** pattern represents the *eXclusive* choice of branches where only one of all possible branches is executed, depending on validity of its guard condition. Its counterpart JOIN (also called Simple Merge) allows to merge all different branches without synchronizing them. **OR-split -join** pattern represents the *inclusive* choice of tasks, i.e., all branches in a inclusive choice whose guard condition is true will be executed. Its counterpart JOIN allows to merge several branches with/out synchronizing them, depending on merge pattern chosen (either Structured Synchronizing Merge or Multi-Merge). **Iteration** represents the multiple executions of one or more tasks.

Semantic Web Service Composition In general, Web services are characterized by providing fine-granularity functionalities. Hence, their application/use are limited for fulfilling specific

requirements. In order to reuse its functionality to produce more complex actions, it is required to combine several web services, which is called *service composition*. Composition languages has arisen to allow composing web services for producing complex web service, of which BPEL [2] (Business Process Execution Language) has been adopted as de-facto standard for this purpose. In essence, these languages define their workflow constructors according to the workflow patterns described above. In the context of web service compositions, processes and tasks correspond to service compositions and individual services (atomic activities), respectively. Thus, a service composition can be seen as a process enjoying all knowledge produced in that research community, particularly those related to the proposed (workflow and dataflow) patterns.

Service composition can be seen as a directed acyclic graph $G = (V, E)$, where each node v_i represents a service selected for execution and each edge e_j represents a control or data dependency between services. More precisely, when a service S_m takes as inputs one or more outputs of a service S_n , it should be executed after, i.e., must be an edge $e = (S_n, S_m) \in E$ in which the services S_n and S_m stand for source and target nodes, respectively.

The semantic web service composition problem can be stated as: given a set of semantic services (as described above) and a request expressed as a 3-tuple $R = \{I', O', E'\}$, find a graph taking only inputs provided in I' and producing all outputs specified in O' . Besides, the effects E' specified in the request must be satisfied by the final state achieved by the composition execution. A basic task during the composition process concerns to decide whether a service produces (some) outputs and effects required by the request or (some) inputs required by the subsequent services in the composition. This task can be accomplish by checking the compatibility of their data (I/O) and conditions (preconditions and effects). These perspectives are not mutually exclusive rather complementary.

Compatibility of I/O is given by the semantic matching notions between concepts used to annotate inputs and outputs, as proposed in [53]. Let two concepts, a provided concept C_P and a required concept C_R , the authors define four semantic matching types: i) *Exact* when the concepts C_P and C_R are equivalent ($C_P \equiv C_R$); ii) *Plugin* when the concept C_R subsumes the concept C_P ($C_P \sqsubseteq C_R$); iii) *Subsume* when the concept C_R is subsumed by the concept C_P ($C_P \sqsupseteq C_R$); and iv) *Disjoint* when the concepts are disjoint ($C_P \sqcap C_R \sqsubseteq \perp$). Although compatibility of I/O allows to increase the degree of automation of discovery and composition tasks, it still suffers of the similar problem like detected in the syntactical description languages (as described above), where different behaviorally services annotated with identical ontological concepts cannot be distinguished.

Along with compatibility of I/O, compatibility of conditions can provide a more precise way for defining the functional semantic of services, where requirements about the current state of world (preconditions) as well as the desired effects after service execution are

taken into account. In order to define how compatibility of conditions are checked, it is fundamental to establish what a state stands for. A state is represented by a knowledge base KB comprising two components: *i) terminological component (TBox)* which defines the terminology of the domain as well its restrictions and, *assertional component (ABox)* which states the facts about the world representing the current state. Thus, to decide whether a condition holds in a state can be boil down for verifying whether it is logical consequence of the current knowledge base, i.e., $KB \models \varphi$ where φ stands for a precondition or an desired effect E' .

The following example illustrates the web service composition problem and how it can be solved. Let both services presented above (PayPal and Online Shop services) and a request R defined according to the Table 3. Analysing each service, it can see that none could individually to fulfill the given request. For instance, Shop service produces all required outputs and desired effects, but its precondition ($\exists hasValidCode.\top(payment)$) is not satisfied by the current state, in other words, it cannot be invoked. On the other hand, whether PayPal service is executed before, it produces the effect ($hasValidCode(auth, \top)$) into the state in order to satisfy the precondition of the Shop service. Thus, whether they were combined sequentially (PayPal ; Shop), all required outputs (Invoice) will be produced and the final state attained will satisfy the desired effect ($hasDeliveryInProgress.\top(invoice)$). Furthermore, all preconditions of the services in the composition are satisfied whenever required (PayPal service precondition trivially holds).

Table 3. Request Example

Element#1	Name#2	Type#3
Inputs	product	Product
	credit card	CreditCard
Outputs	invoice	Invoice
Effects	delivery	$\exists hasDeliveryInProgress.\top(invoice)$

3 Automatic Semantic Web Service Composition Approaches

This survey considers only approaches providing support to the automatic generation of semantic web service compositions, i.e., those approaches attaining the composition model from a given composition request. Were excluded from the analysis approaches demanding as input some form of abstract composition model, in which case the problem boils down to discovery services in order to instantiate an executable composition model. Also were not considered approaches generating semi-automatically or manually the composition models. By semi-automatic it means those approaches demanding some form of user/developer intervention during the generation of the composition model, for instance, the decision of which

workflow constructors should be included in the composition model.

This section presents a brief overview of a number of techniques employed for automatic Semantic Web Services (SWS) composition, including a brief description of some relevant works for each technique. It also presents a comparative table showing the supported workflow patterns for each approach.

The classification of the approaches used in this paper follows that proposed in [8], namely: (i) *AI Planning*, (ii) *Mathematical-based* and (iii) *Model-based*. This classification was adopted because it is the most complete and comprehensive among all those found in the literature. A fourth category was included on the classification, grouping some works that do not fit in the aforementioned categories.

3.1 AI Planning Techniques

AI Planning techniques have been widely explored in the literature, to generate solutions for the automatic SWS composition problem. This paper follows the categorisation of AI planning techniques proposed in [23], which is one of the most comprehensive taxonomy of composition methods in the AI planning literature.

3.1.1 Classical Planning A planning problem P is defined as a triple $P = (\Sigma, s_0, g)$, where Σ is the planning domain, s_0 is the initial state and g corresponds to a set of goal states. The planning domain is given by a state-transition system $\Sigma = (S, A, \gamma)$, where S is a set of states, A is a set of actions and $\gamma : S \times A \times S$ is the state-transition relation (or function). The structure of solution plan for P is dependent of the planning technique employed, as well as of the assumptions¹⁰ about the planning domain Σ . Classic Planning techniques are characterized by the proposition of solution plans which are given by restricted state-transition systems. This assumption means that the planning domain Σ is *static*, *deterministic*, *finite* and *fully observable* (we refer to [23] for a complete discussion).

Classical Planning techniques are divided into two main branches: State-Space Planning and Plan-Space Planning. These branches are characterized next.

State-Space Planning This form of classical planning techniques characterize themselves by represent the states Σ as a set of ground atoms. The goal of the planning is to find a *sequence of actions* which, when applied to the initial state, it leads to a goal state. Formally, a solution of P is a sequence of actions $(a_1, a_2, \dots, a_k)$ corresponding to a sequence of state-transitions $(s_0, s_1, \dots, s_k)$, such that $s_1 = \gamma(s_0, a_1), \dots, s_k = \gamma(s_{k-1}, a_k)$, and s_k is a goal state. Search can be done considering both directions: (i) from initial state to goal state

¹⁰Static vs. Dynamic, Deterministic vs. Nondeterministic, Finite vs. Infinite, Full vs. Partly Observable, Implicit vs. Explicit Time

(forward) and (ii) from goal state to initial state (backward).

Plan-Space Planning These techniques use a different search space than state-space planning. Plan-Space Classical Planning is characterized by having *partial plans* as search space nodes, in which actions are only partially ordered and partially instantiated. Transitions between nodes of the search space represent operations of plan refinement, namely, add an action, add a causal link, add an ordering restriction and, add a variable binding. Planning starts with an initial plan composed by two actions a_0 and a_∞ (called dummy actions), where the action a_0 is only equipped with effects corresponding to the initial state (without preconditions) and the action a_∞ is only equipped with preconditions representing the goals (without effects). Besides, a single ordering restriction $a_0 \prec a_\infty$ is added to the initial plan. This initial plan is successively refined until no flaws exists (success) or no more operation is applicable (failure).

The PORSCE II framework described in [30] models the semantic web service composition problem as a planning problem, expressed in PDDL¹¹, and then employ a PDDL-compliant planner to get a solution plan for achieving the user goals. PORSCE II is responsible for assembling/preparing the planning domain and problem from the available set of services descriptions and domain ontologies as well as from the user requirements. Firstly, PDDL planning domain is formed by translating each OWL-S service (profile) description WSD_i to an action A_i . Union of the inputs and preconditions of the OWL-S service profile defines the preconditions of the action A_i , whereas union of outputs and positive effects¹² of the service define the list of positive effects of the action A_i . The list of the negated effects of the action A_i is given in a similar way, however considering only union of the negative effects. On the other hand, user requirement is used for forming the planning problem. PORSCE framework is also responsible for providing support for semantic analysis (in form of hierarchical relationships and semantic distance) aiming to enhance both planning domain and planning problem with similar and equivalent concepts to those used to describe the services profiles and the user requirements. After assembling the planning domain and problem, a PDDL-complaint planner is invoked in order to produce a plan containing actions structured with sequential and parallel constructors, which is translated to an OWL-S process model.

In [70], the authors propose adaptations to the Partial Order Planning technique called SC-APOP, by extending it in order to support actions with multiple conditional effects. Pre-conditions and (conditional) effects are expressed by a conjunction of predicates. The conditional effects are represented as a pair of conditions and effects. The paper does not highlight whether domain ontologies and semantic reasoning is employed in the composition process.

¹¹Planning Domain Definition Language [22].

¹²A positive effect represents a new fact to be included in the current state, whereas a negative effect represents an existing fact that must be retracted.

Partial order plans generated by SC-APOP planner are translated to a directed acyclic graph (DAG) $\langle V, E \rangle$, where each $v_i \in V$ is a vertex corresponding to an action step and, each $e_i \in E$ is an edge corresponding to an ordering constraint. After translation, it is performed a *Transitive Reduction* [1] on the DAG aiming to reduce the complexity of the workflow diagram to be generated (while preserving the execution flow). Then, in the post-processing phase, the final workflow diagram is produced from the reduced DAG by adding new vertices to represent the recognized workflow patterns, which are recognized by analysing the structure of each vertex in the reduced DAG.

It is interesting to note that some of the rules presented in [70] are identical to each other (for example, those corresponding to the parallel-split and exclusive-choice patterns). Moreover, it is not clear from [70] whether some workflow patterns, such as *simple-merge*, *multiple-merge*, *multiple-choice* and *synchronization-merge* are covered by the tool. We have not considered them in our analysis.

3.1.2 Neoclassical Planning Neoclassical Planning techniques are intended to provide new representations for the search spaces as well new algorithms to explore them. It aims to increase the capabilities for solving more complex problems than those managed by the classical planning techniques. The main difference between classical and neoclassical techniques concerns that a node represents in the search space. In classical planning, each node of the search space represents a partial plan meaning that all actions contained in it will be required to appear in the solution plan. In the other hand, in neoclassical planning each node represents a set of partial plans meaning that not all actions contained in the set of partial plans will be included in the solution plan.

Neoclassical planning techniques can be divided into two branches: *Graph-plan* and *Satisfiability*.

Graph-Plan This technique introduces a very powerful search space called *planning graph*. Its main characteristic is the existence of a multi-level structure, which intercalates a propositional level with an action level. The first level of the planning graph is given by a propositional level representing the inputs of the planning problem. Then, the planning algorithm generates an action level containing all applicable actions for that proposition level, as well as a new proposition level containing all previous facts together with the effects of the actions in the action level generated. This phase is called *graph expansion*. After each graph expansion, the planning algorithm executes a phase called *solution extraction*, in which it checks whether all required outputs are present in the last proposition level and, if true, it executes a backward search to extract the solution plan. This cycle is repeated until a solution plan is extracted (success) or no more actions can be applied in order to expand the graph (failure).

In [37], the authors present an approach to tackle the semantic web service compo-

sition problem by extending the AI planner XPlan [32], which is an heuristic hybrid search planner based on the FF-planner. Its XML parser module is responsible for parsing the domain and problem definitions and for compiling them into memory efficient data structures. In the preprocessing module, relevant facts and relevant operators are identified among all possible operator instantiations. In this module, both relevant facts and operators are used to build a graph which represents the dependencies between the precondition, add- and delete lists of operators, and facts. In the core (re-)planning module is applied the graph planning algorithm along with two heuristics, namely (i) relaxed graph plan to establish the distance between an initial state $\mathcal{I}$ to all its reachable states $\mathcal{S}$ (these distance values are then used to guide the forward directed search) and (ii) enforced hill-climbing search for searching for the best reachable states during the generation of the global plan. From the derived plans, it is possible to identify/extract composite services with two workflow patterns, namely sequence and split-join.

The author in [6] proposes a composition process consisting of three basic steps. The first and second ones look for services consuming the provided inputs (in the composition request) and services providing the requested outputs, respectively. However, in most of the cases, only these (initial and final) services are not sufficient to provide the overall requested functionality, so requiring to discovery and compound intermediate services. In the third phase, a forward chaining strategy based on graph planning algorithm is applied in order to select and compound usable services. The solution is extracted applying a backward search strategy for each input of the services appearing in the final actions layer. The resulting plan comprises sequential and AND-split constructors.

Satisfiability The main idea of the techniques falling in this category of Neoclassical Planning is to formulate the planning problem as a satisfiability problem, i.e., as the problem of determining whether a propositional formula is satisfiable. After coding the planning problem as a propositional formula, a decision procedure is employed in order to assign truth values to the propositional variables so that the satisfiability of the propositional formula can be determined. Then, from the truth values assigned to the propositional variables, a plan can be extracted.

The authors in [52] present a monolithic approach to generate service compositions by separating the composition and execution phases. In that work, the semantic web service composition problem is formalized using axioms of the Event Calculus [62] and use an Abductive Theorem Prover [63] to generate the solution plans. OWL-S services descriptions and user inputs/outputs are converted to axioms of the Event Calculus. Based on these axioms, ATP generates a narrative comprising a sequence of time-stamped events, whereas the residue keeping a record of the narrative represents the derived plan. A plan is constituted by a list of happens and before predicates representing service invocations and ordering restrictions, respectively. The generated plan is translated to a graph whose set of vertices is represented

by the happens predicates and the set of edges is extracted from the temporal orderings given by the before predicates. Lastly, the graph is processed in order to recognize sequential and split-join constructors and generating the final composition expressed as a process model of an OWL-S service.

The work presented in [36] has similarities with the previous work [52], i.e., both take the Event Calculus framework as basis to tackle the automatic service composition problem. A main difference between these two proposals is that in [36] the composition and execution tasks are interleaved. The proposed framework translates each OWL-S service profile for an event description in the event calculus framework, whereas input types and values supplied by user are translated to event calculus axioms (by an *initially* predicate) to describe the initial state of the world. As result, the ATP can generate both concurrent and sequential set of events, depending on time-stamps of events assigned to the list of happens and before predicates. Because it is an interleaved approach, the authors considered only information-providing services, thus avoiding to treat with transactional execution framework.

3.1.3 Planning with Heuristics and Control Strategies This class of planners is characterized by using knowledge about the application domain, as well as some kind of control strategies to subsidize the planner with domain information for the generation of the solution plan. The main motivation of these strategies is to leverage the knowledge of domain experts in order to help on the plan generation process. Among the techniques belonging to this sub-category, Hierarchical Task Network (HTN) is the unique explored in the literature for generating web service compositions.

Hierarchical Task Network (HTN). HTN (Hierarchical Task Network) is a planning technique that exploits the built-in hierarchical structure of the problems in order to generate solutions for them. The central idea of this technique is to decompose a complex problem into sub-problems successively until each sub-problem cannot be further subdivided and can be solved by a primitive action. An HTN domain comprises two elements: primitive and non-primitive actions (also called tasks). The key point of using this technique is to correctly define the non-primitive actions, which allows exploiting the knowledge of the domain experts. In the web service composition context, each available web service is taken a primitive action.

In [64] it is presented an HTN-based approach to tackle the web service composition problem. More specifically, SHOP2¹³ planner is applied in order to generate the composite services. The approach exploits the similarity between the OWL-S process model and HTN methods, where OWL-S atomic and composite processes are translated to HTN primitive actions and methods, respectively. In this approach, the parallel constructs of the OWL-S

¹³SHOP2 is a domain-independent HTN planning system [64].

process model are excluded from the translation process, since that HTN techniques do not have support for concurrent actions. The solution plan generated by the planner is composed by a sequence of actions (OWL-S atomic process).

The authors in [60] address the service composition problem as a graph heuristic search problem. Firstly, a dependence graph is generated based on the available services and user request. After, it is reduced by eliminating unused services and combining the equivalent ones. From the reduced graph, the method applies the A^* algorithm to find out the best path, i.e., a path that minimizes the number of services and steps to achieve the desired solution. Because the graph is arranged in layers, resulting service compositions can include *sequential* and *parallel* constructors.

The authors in [48] present a heuristic search algorithm called Enforced Hill-Climbing for automated service composition, supporting the creation of compositions with AND-Split and OR-Split¹⁴ constructs, allowing uncertainty about the initial state and service effects.

Summary of this section Before we continue the analysis of other methods, we can summarize the results so far by noticing that the problem of automatic generation of service compositions was also treated by a number of AI techniques, taking advantage of the wide range of knowledge and tools produced by that research community. This is due to the fact that the automatic service composition problem has many similarities to those considered in the AI planning community. However, despite the similarities, the service composition problem has particularities which are not considered by AI planning community, as for example, using of domain ontologies for expressing the semantic of the web services. Thus, as it can be observed in the surveyed works, most of them employ a pre-processing phase in order to enhance the (AI planning) domain with *semantically equivalent* informations (from domain ontologies) before invoking an AI planner to generate solutions (compositions). Another aspect that is worth to be highlighted, it is that generated composition models are structurally limited by the AI planning technique employed, i.e., it is not possible to swerve from the limitations imposed by each AI planning technique.

3.2 Mathematical-based Techniques

This section reports some works that employ mathematical structures and techniques to model and generate web service compositions.

3.2.1 Graph-based Techniques Graph-based approaches explore well-established graph theory for modelling and generating Web service compositions. In general, the Web service

¹⁴Invoking a service with uncertain effects results in several possible states. If subsequent services cannot be invoked in all states, an OR-split is added to the composition.

composition problem is modelled as a graph where nodes represent the services and edges represent the relationship between their inputs and outputs. Based on the I/O dependencies between services modelled by the graph, the composite service is extracted by applying search algorithms.

In the work presented in [66], the authors propose a semantic web service composition approach in which both discovery and composition phases are based on a graph structure called *semantic network*. Nodes of the semantic network represent semantic Web services and edges represent data dependency between services. Each edge is labelled by a value which represents a degree of similarity between the output of the outbound service and the input of the service to which the edge incides. The degree of similarity is given by the subsumption relationship between concepts assigned to the outputs and inputs, which is defined as *Exact*, *Subsume*, *Plugin* and *Disjoint* [53]. From the request's outputs, the algorithm carries out a search in the semantic network for all the services whose outputs are similar to the requested outputs. Then, for each found service is created a composition plan. Each of the inputs of these services are treated as new goals to be solved, i.e, a new search is carried out for discovery services that satisfy these new goals. This search is repeated until no more outputs or goals to be satisfied. Resulting plans are orchestrated with sequence, AND-Split and AND-Split -Join control constructs.

DynamiCoS framework presented in [15] aims to provide mechanisms to tackle each of the phases and requirements of the service composition life-cycle. Services are published along with their semantic annotations in a representation format which is agnostic in regarding to the description languages and technologies. In that formalism, services and service compositions are respectively represented as *tuples* and *graphs*. Elements in tuples are assigned to concepts defined in domain ontologies allowing exact and partial semantic matches in the service discovery. DynamiCoS organizes the descriptions of the discovered services in a preprocessed structure called Casual Link Matrix (CLM) [40], which stores all possible semantic connections (or causal links) between the discovered services input and output concepts avoiding to perform (computationally expensive) semantic reasoning during the composition process. Based on CLM, a graph-based composition algorithm tries to find a service combination that fulfil the service request. When it exists, a composition containing sequential and AND-split constructs is returned.

The work presented in [38] formalizes the composition problem based on a conditional directed acyclic graph. Differently of other approaches employing graph, in that formalization a node stands for a service or postcondition of the immediate predecessor service which can only be evaluated at runtime. Thus, it can express the conditional constructors. Based on this graph, a two-phase algorithm is applied in order to generate the composition graph. The first phase is concerned with forward graph construction considering the query input parameters and service repository, which employs a filtering technique to select useful

services at each stage of the graph construction. Each layer of the graph contains all applicable services considering the available inputs (given by union of the services outputs in the previous layer and the request inputs). Graph building continues until all query outputs and postconditions are produced by the services in the composition. In order to avoid cycles, the method only take into account those services which require at least one parameter from the outputs produced in the previous stage. In the second phase, an analysis is carried out in the reverse direction (backward) in order to remove redundant services which do not directly or indirectly contribute to the query output parameters. Then, the conditional DAG is translated into a OWL-S process model, which includes sequence, AND Split-Join and XOR-Split constructs.

The authors in [56] propose a technique for the automatic composition of semantic web services exploiting a weighted graph-based model. They divided the composition process in three steps. Initially, the semantic web services repository is converted to a graph where each node represents a service and each edge represents a connection between services having some semantic similarity among their inputs and outputs. Semantic similarity is computed by a semantic reasoner based on the subsumption relationship between concepts used to annotate the inputs and outputs. This graph is updated off-line as services are included or excluded from the repository, independently from any composition request. Secondly, at the time when a service composition is demanded, the graph is enhanced with extra *virtual* nodes representing the request inputs and outputs. In this step, the weights initially assigned to the nodes of the graph are adjusted by applying a policy of discount, which takes the service inputs and outputs that has been provided in the request parameters into account. Initially, the weight of each node was arbitrarily fixed as three times the number of inputs and outputs of respective service, i.e., the cost for using the service depends on its number of inputs and outputs. Based on that graph, the authors use Dijkstra's shortest path algorithm to find the path with minimum costs. Since the algorithm computes one shortest path for each output, some paths may present some intersection. The algorithm, together with cost policies, generate partial compositions as paths. In the third step, the different paths generated are merged to produce the final composition. From the merged graph, is possible to extract service compositions with sequential and AND-Split constructors.

The work presented in [13] follows a similar idea to the other works in this category. It structures the available set of semantic web services as a graph, where nodes represent services and edges represent connections between services having some semantic relationship between their inputs and outputs. In their proposal, nodes are called service cells and the graph contains two kinds of nodes: ordinary and special nodes, representing a service operation and the request parameters, respectively. The algorithm solves the composition problem in four main stages: preliminary stage, search stage, simulated execution stage, and selection and execution stage, where only the first two are relevant to our analysis since they are related to the composition task. The preliminary stage is concerned with the inclusion of

special nodes into the (initially preprocessed) graph for representing the request parameters. Specifically, are included as many endSC special nodes as the request outputs. Also, it is included a unique globally-accessible special node called startSC, representing the missing input parameters to be provided by the user, i.e., for each service cell having some input missing, there will be a link connecting this node with the special node startSC. Based on this graph, all sub-graphs satisfying the user request are searched for. The algorithm performs a breadth-first search (from the special nodes endSC backward to special node startSC) in an AND/OR tree, where an AND node is an service cell for which all its inputs have been provided, whereas an OR node is one input which has to be provided by at most one output of another service cell. This work has an interesting feature that differentiates itself from the other works: it does not require that request inputs are provided in advance by the user (which only specifies the requested outputs). From the request outputs the missing inputs are calculated and shown to the user.

3.2.2 Logic-based Techniques This section presents some works that employ logic reasoning to generate service compositions. In general, logic-based approaches use *proof-as-program* methods [9] for such a task. In these methods, the available set of services and service requests are translated, respectively, into axioms and theorems of the underlying logic. An automated theorem prover is used to find a proof for the theorem from which the program is extracted.

Both works presented in [57, 54] model the semantic web service composition problem using Linear Logic (LL) [24], and employ a LL theorem prover to produce proofs from which are extracted the solutions (i.e., the composite services). Also, both works use process calculus [50] to represent the composite service generated from the complete proof.

In [57] it is presented a method for the automated Web service composition, which is based on proof search in a fragment of propositional Linear Logic. Firstly, each semantic description of the available Web services (in the form of DAML-S Service-Profile) is translated into a extralogical axiom of LL ($\Delta_c \vdash ((I \otimes P) \multimap (O \otimes F) \otimes E) \otimes \Delta_r$), while the requirement to a composite Web service (including functionalities and non-functional attributes) is expressed in form of a LL sequent to be proven ($\Gamma_s, \Delta_c \vdash ((I \otimes P) \multimap (O \otimes F) \otimes E) \otimes \Delta_r$). An LL sequent representing a service request can be explained as: find a combination of services (given by Γ_s) allowing to compute the required outputs O from the provided inputs I and that also transforms an initial world satisfying P into another one satisfying F . For this task, an LL theorem prover is invoked to check whether the request can be satisfied by a composition of existing services. In order to decide whether a service is useful in a proof, a semantic reasoner can be invoked for checking subtype relationships between propositional variables used in the axioms and theorem. To directly extract the process model for the composite service from the generated proofs, a process calculus is attached to the LL inference rules in the style of type theory. After this step, the generated process models can be trans-

lated to BPEL or OWL-S process model, based on an upper ontology proposed to represent process models.

The work presented in [72] explores the use of theorem provers to generate solutions for the semantic web service automatic composition problem. Specifically, the theorem proof approach employed by authors is based on the deductive program synthesis theory¹⁵ [44]. In their proposal, the functional specification (given by inputs, outputs, preconditions and effects) of all the semantic web services are translated into first-order logic formulas, based on following template:

$$\begin{aligned} \forall x_1, x_2, \dots, x_n (T_1(x_1) \wedge T_2(x_2) \wedge \dots \wedge T_n(x_n) \wedge P(x_1, x_2, \dots, x_n) \\ \Rightarrow (\exists r (T_r(r) \wedge Result(x_1, x_2, \dots, x_n, r)))) \end{aligned}$$

where $x_1, \dots, x_n$ represent the inputs of the service, r represents the output of the service, $T_*(k)$ means that parameter k has the type of T_* , $P(x_1, x_2, \dots, x_n)$ means that $x_1, \dots, x_n$ satisfy the precondition P which is defined in the knowledge base. Each request submitted to the composition system is encoded as an object formula. Based on axioms representing the available services (according to the above template), the composition system employs the theorem prover OTTER [35] for proving the object formula. If the proof exists, the implementation of a composite service containing sequential and parallel constructs is extracted from it.

The use of Description Logic (DL) [3] as a formalism for modelling the service composition problem was originally proposed in [4]. In that work, services and service compositions are formalized within DL to provide a well-defined semantics. A service S is defined as a pair $S = (P, E)$, where P is a finite set of ABox assertions representing the preconditions, and E is a finite set of pairs φ/ψ representing the conditional post-conditions. Based on that service representation, the formal semantic for service is defined as a *transition relation on interpretations*, in which a service S may transform an interpretation I in another interpretation I' (denoted as $I \Rightarrow_S^{T,A} I'$). Furthermore, two important reasoning tasks about services are defined, namely *executability* and *projection*. In [42], these reasoning tasks are used in order to define the notions of service substitutability, such as, identical service and substitute service. Then, an algorithm based on backward-chaining strategy is employed to reason about how to compose services to achieve an user-defined goal. It divides the goal in two parts, where the first one encompasses the first ABox assertion (contained in the goal) and another one encompasses the remainder of the goal, and then search for services that satisfy the first part. Thereafter, it executes a new iteration taking as goal (for this iteration) the remainder part of the previous goal. The algorithm terminates when some goal could not

¹⁵Deductive program synthesis theory is based on the observation that proofs are equivalent to programs: Each step of a proof is interpreted as a computation step and the organization of the proof induces the use of different program constructors.

be satisfied by the available services (fail) or when all unsatisfied goals correspond to the assertions in the initial goal (success). The service compositions generated by the algorithm includes the sequential and parallel constructors.

The work presented in [59] proposes a method containing rules for (i) encode OWL-S atomic process into a set of rules expressed in the ontology-aware rule language SWRL [34], and (ii) apply a backward search planning algorithm to compose them. The rules considered in that work are *safe*, i.e., only variables that occur in the antecedent of a rule may occur in the consequent. The composition algorithm employs a backward search strategy to accomplish the composition task. It takes as input a knowledge base containing SWRL rules and a goal specified as a SWRL atom, and it returns every possible path built combining the available SWRL rules in order to achieve the goal. From these resulting paths, is possible to extract a sequence of rules application representing the (sequential) composite service that satisfies the requested goal. The authors do not mention the criteria used to ranking the sequence of rules in order to select the best one.

3.2.3 Techniques based on Process Algebra Process algebra is a family of formalisms that allows the automatic verification of certain properties of process behaviours. Their underlying semantics is based on Labelled Transition Systems (LTS). They provides a technique to verify whether an implementation matches a given specification. This technique is based on the notion of on bi-simulation [49], which aims to check the behavioural equivalence of processes. In the context of web service composition, this verification is useful to decide whether a service can be substituted by another one through analysing their behaviour.

The work presented in [20] employs a model checking approach to tackle the automatic semantic web service composition problem. Services and their dependences are modelled as a set of interleaved processes in a process algebra, specifically, in CSP (Communicating Sequential Process) [31]. In the first step, each service having more than one result is transformed into multiple services, each of which having exactly one result. Then, for each input/output variable of each service, it is defined a boolean variable to keep track of its availability and so facilitating the service composition process. Its purpose is to indicate (setting it as true) whether its value is passed (produced) by some output of other services. Similar idea is applied to service output variables, however, indicating whether some service output is passed to some input of other service. Intuitively, the behaviour of each process representing the available web services can be defined as: when all its inputs become true, the process executes an internal event τ and then sets all output variables to true (indicating that the service outputs were produced). Similarly, services dependencies are also modelled as process. In the final stage of the composition method, all processes representing services and services dependencies are put together by using the sequential and interleaving operator, and then it is checked the non-terminating property of this resulting interleaved system. If the property cannot be proved, a composition graph is constructed from event traces of the

counter-example generated by the model checker (otherwise no composition is possible). The workflow patterns supported by the approach are *sequence* and *parallel with synchronization* constructors.

Summary of this section Surveyed works in this section exhibit a number of possibilities for tackling the service composition problem. Works modelling the composition problem as a graph present a higher level of flexibility since the meaning assigned to the nodes and edges are system-dependent, i.e., it is not fixed. For instance in the work presented in [38], nodes are treated as services or postconditions, which allowed increasing its expressiveness by supporting the XOR-split pattern. Works employing theorem proving are more restricted in their use, i.e., they must work according to specific logics and theorem provers (much alike those works employing AI planning techniques). During the literature review, a few works were found using process algebra in order to automatically obtain service compositions.

3.3 Model-based Approaches

This category groups approaches that represent web services and service composition by using a higher-level description on top of the traditional service description in WSDL, OWL-S or any other similar framework.

3.3.1 Approaches that use Petri Nets Petri net is well known for its ability to model concurrent systems. Its underlying model is represented by a directed bipartite graph, whose nodes are divided in two disjoint sets: places and transitions. Places can store a number of tokens, whereas transitions represent state transformations. Triggering a transition depends on current net configuration, i.e., whether its precedent places have a sufficient number of tokens to enable its activation. When a transition triggers, one or more tokens in the precedent place(s) are transferred to the subsequent place(s). These connections between places and transitions are given by the arcs of the graph. Some of the works that use Petri nets to help on the automatic web service composition are described next.

The work presented in [67] proposes a method for semantic Web service automatic composition based on logical inference of Horn clauses and Petri nets. In that work, dependency relations between services are transformed in dependency rules expressed as Horn clauses, thus forming a rule base on which logical inferences are carried out. These dependency rules are (pre-)computed off-line before any composition query to be submitted. They have the following structure: $ws_1 \wedge ws_2 \wedge \dots \wedge ws_m \rightarrow ws_k$, which indicates that when all services ws_j ($1 \leq j \leq m$) are executed, then ws_k can be invoked. Furthermore, some constraints are applied in order to ensure the validity of the rules, e.g., that all inputs of service ws_k are provided by some of the outputs of the services ws_j ($1 \leq j \leq m$). Similarly, the input/output query is also modelled as rules in Horn logic, where the input is expressed

by a fact ($\rightarrow ws_p$) and the output is expressed by a goal ($ws_q \rightarrow$). Then, based on both rule bases (service dependency and query-specific), an algorithm employing a *forward chaining strategy* for propositional logic is applied in order to determine whether the composite service exists. The inference problem can be formulated as determining whether there exists one composite service that can satisfy a client query can be done by determining whether ws_q is entailed by $KB \cup \{ws_p\}$, or by determining whether $KB \cup \{ws_p\} \cup \{\neg ws_q\}$ is unsatisfiable. As result it returns a rule set necessary for the composition that can satisfy a composition query. Thus, given a set of Horn clauses as $KB \cup \{\rightarrow ws_p\} \cup \{ws_q \rightarrow\}$, the incidence matrix of a Petri net corresponding to the set of clauses is obtained by applying the procedure proposed in [51]. From the Petri net representation, structural analysis techniques (such as T-invariants) are used to obtain the composite service, which can contain *sequential* and *parallel* constructors.

The authors in [71] propose a method based on Petri Net coverability to handle automatic web service composition. Every input and output of a single service are mapped into places of a Petri net, whereas each service is treated as a transition of a Petri net. A token in a given place means that the corresponding input was provided in a user request. According to user's input and desired output, the initial and target markings of the net can be obtained and the coverability graph can be constructed. Coverability graph is exploited in order to figure out all shortest paths covering the final marking and then extracting a sequence of service executions from the transitions.

3.3.2 Finite State Machines Finite State Machines [33] are ubiquitous in Computer Science. They provide a simple and versatile model of computation that can be both treated by humans and algorithms. Among the techniques that use Finite State machines for the automatic generation of web service compositions we can cite:

In the framework presented in [10], the service behavior is described in terms of its possible action executions (execution trees) represented by Finite State Machine (FSM). Service behavior is represented by a schema reflecting two different perspectives: user perspective (given by the External Tree) and composer perspective (given by the Internal Tree). From this setting, the problem of composition synthesis can be translated into a problem of synthesizing an internal schema for a service E which is a composition of E regarding to the community (repository) of services.

Other work that exploits Finite State Machines is presented in [29]. It employs Interface Automata [16] as a formalism to model web services, exploiting its advantages/features with regard to the composition concept, which allows that the composition of two components can be calculated. In order to solve the composition problem, the available services (represented as interface automata) are arranged in a dependency graph in which nodes represent the inputs and outputs of web services and edges represent the associated web services.

Then, given a composition request (also in the form of an interface automaton), it is proposed an algorithm encompassing two steps: (i) find the potential web services and (ii) find a composition model that satisfies the given request. As result, the algorithm produces a binary expression tree representing the composition model that satisfies the request, where leaves represent (atomic) services and intermediate leaves represent workflow constructors (*conditional*, *parallel* and *sequential*).

Summary of this section Approaches in this category exploit model-based techniques in order to tackle the service composition problem. The basic idea of this family of solutions is to represent the available services and the composition requests as a particular model and then employ techniques that allow to reason about it in order to extract the compositions. In Petri nets models, places represent the inputs/outputs whereas transitions represent the services. Model checking based approaches exploit the representation of services and their dependencies as process in some process algebra. Then, the composition request is expressed as a property to be verified against the given model. Whether the property holds, the service composition is extracted from the traces returned by model checker.

3.4 Other Approaches

The works reported in this section do not fit in the aforementioned categories and represent other approaches for the generation of service compositions. The semantic web service composition problem has also been tackled with techniques from several areas of computing, such that database techniques and evolutionary computing.

3.4.1 Database Techniques In this approach, the idea is to reduce the service composition problem to a query answering problem, and to employ query rewriting techniques to obtain service composition solutions. The motivation for answering queries via query rewriting is the prospect of using existing deductive database systems to evaluate the computed rewritings, thus taking advantage of the query optimization strategies provided by such systems.

The work presented in [5] proposes an approach to treat the composition problem of data-providing services, i.e., services that do not produce effects that change the state of the world, only retrieving data. This kind of service cannot be unambiguously expressed in the traditional conceptual models of semantic web services, such as OWL-S and WSMO, since that functional semantics depends on the specification of the service effects. That is, given services having similar inputs and outputs (though being described by ontological concepts) it could not be possible to distinguish them by their functional semantic. In that paper, the authors propose to model data-providing services as RDF views over a mediated (domain) ontology, where each RDF view contains concepts and relations from the mediated ontology

in order to capture the semantic relationships between input and output parameters. From the set of RDF views (representing the available data-providing services), the framework accepts as input a query expressed over a mediated ontology and then applies a query rewriting algorithm in order to answer the user query based on these RDF views, transforming automatically the user's query (during the query rewriting stage) into a composition of data-providing services. The generated composition contains services orchestrated with sequential and parallel constructors.

The approach proposed in [46] combines query rewriting and configuration to provide a semantic-based approach to generate service compositions. At the first phase, called discovery phase, a rewriting query algorithm is employed in order to decompose the user query and the set of service descriptions into sets of services that implement the required functionalities. It relies on a modified version of the bucket algorithm presented in [28]. Configuration is used for generating partial orders for the selected services observing the user preferences and the different levels of business rules (composition, service and user). Resulting composition allows to perform independent services at the same time (i.e., in parallel).

The work presented in [41] addresses the service composition problem with a different database-based approach from other works in this category. The proposed system, called PSR system, pre-computes the web service compositions using joins and indices. It also supports semantic matching. The composite service generated by this technique may include sequential and parallel constructors.

3.4.2 Evolutionary Computing Other research area that has contributed to generate solutions for the SWS composition problem is Evolutionary Computing [18].

Several works use this class of techniques to tackle the automated composition of services. Among them, [69, 11, 43] apply Genetic Algorithms to achieve optimal compositions with regard to QoS aspects. All these works assume that the workflow has been previously generated or manually created. One exception to this is the work presented in [61], which employs Genetic Programming [39] to generate semantic web services compositions. In order to be able to generate the (prospective) solutions, the algorithm relies on context-free grammars to build the valid structures of the composite services (chromosomes), i.e., the types of composite services that are going to be build. A chromosome represents a possible solution (composite service), where each intermediate node of the tree represents a workflow constructor and each leaf represents an atomic process. Each node of the tree includes some attributes used to provide necessary informations to guarantee the validity of the chromosomes, which are updated when crossover and mutation operations are performed. Moreover, the proposed method aims to generate solutions with the fewer number of services as possible. The solutions generated by the approach contain a number of workflow patterns, namely, sequence, parallel, parallel with synchronization and choice (as definition of the context-free

grammar).

4 Discussion

The analysis presented in this survey takes as basis the basic workflow patterns [68], as described in the section 2. Table 4 presents an overview of the workflow patterns supported by each work presented in the previous section. The workflow patterns presented in the table are arranged in increasing order of complexity (sequence, AND-Split, and so on), so that the reference indicated on each column of the table corresponds to the most complex workflow pattern supported by the work (consequently, the workflow patterns of the precedent columns are supported too).

Table 4. Workflow patterns generated by each work.

Category	Sub Category	Sequence	AND-Split -Join	XOR-Split -Join	OR-Split -Join	Iteration
AI Planning	<i>Classical</i>		[30] [70] ²			
	<i>NeoClassical</i>		[36] [37] ¹ [52] [6]			
	<i>Heuristics & Ctrl Strategies</i>	[64]	[60]	[48]		
Math-based	<i>Graph-based</i>		[66] [15] ¹ [56] ¹ [13]	[38] ¹		
	<i>Logic-based</i>	[59] [72]	[42] ¹	[57] [54]		
	<i>Process Algebra</i>		[20]			
Model-based	<i>Petri-net</i>	[71]	[67]			
	<i>Finite State Machine</i>	[10] ³		[29] ¹		
Others	<i>Database Techniques</i>		[5] [46] [41]			
	<i>Evolutionary Computing</i>			[61] ¹		

¹Join pattern not supported or not explicitly reported in the paper.

²As aforementioned, not all workflow patterns are clearly supported by this approach - there are identical rules.

³The text does not make clear which are the supported workflow patterns.

The discussion presented in this section is based on observations done during the literature review. Its very difficult to quantitatively evaluate the corpus of work, due to the diversity of tools, languages and techniques employed, as well as because of most of the authors do not make their implementation publicly available.

Owing to the similarity of the service composition problem and the AI planning problems, several techniques and algorithms coming from the latter research area have been em-

ployed/adapted to generate solutions for the automatic web service composition problem. During the literature review were found papers employing almost all existent AI planning techniques to automatically generate web service compositions, except perhaps, for the state-based planning technique. In particular, the graph-planning technique has been extensively employed in the surveyed works of this category. The widespread use of this technique may be explained by its flexibility to combine with others techniques (such as HTN [37]), as well as to the use of heuristics [60], aiming to increase the performance of the proposed algorithms.

From the viewpoint of the supported workflow patterns, AI planning shows some known limitations, that are inherent to these techniques. Both works [30] and [70], classified here as classical AI Planning, employ plan-space planning technique for generating the plans, which are represented by a sequence of partially ordered actions. This sequence can be exploited in order to detect independent actions which can be executed in parallel. Techniques classified here as neoclassical, in particular graph-planning, also provide support for generating plans encompassing the execution of parallel actions. In graph-planning, the parallelism can be obtained by exploiting the set of actions contained in each action layer, i.e., if there are multiple actions at the same layer, then they do not have dependency between each other and can be executed in parallel. In the subcategory Heuristics and Control Strategies, the decomposition method employed by the HTN technique imposes the limitation of generating only sequential plans, as observed in [64]. One work that stands out in this subcategory is that presented in [48], whose supported workflow patterns include the eXclusive OR (XOR). However, the higher expressiveness level observed in that work is not due to the employed heuristics, but to the fact that the authors consider each node of graph (that represent the composition) as a set of states, rather than as a single state.

The approaches belonging to Mathematical-based category present, in general, a similar expressiveness level to that observed in the latter category, i.e, extensive support to sequence and parallel constructors. With regard to the graph-based techniques, their support of the parallel constructor is due to the backward chaining strategy employed for generation of the compositions, where independent services partially fulfilling the goal (required outputs) are orchestrated in parallel. One work that exposes a higher expressiveness level (including the XOR pattern) in this category is that present in [38]. Its support to the XOR pattern is due to meaning assigned to the graph nodes, which can represent services (as usual) in addition to conditions that are evaluated only at runtime. Approaches employing Logic-based techniques show a lower expressiveness level, when compared to the previous technique in this same category, in general, providing support only for sequence constructor. The method proposed in [42] differs from others in this subcategory due to employ an AND-OR tree-based structure to represent the compositions, where OR branches represent different ways to achieve a goal and AND branches represent arrangement (combination) of services used to achieve a goal. The works presented in [57] and [54] stand out in the Logic-based subcategory

by provide support to XOR pattern.

There are few works belonging to the Model-based category when compared to the number of works in the previous categories. There are other works beside those mentioned in this survey, but their focus is to employ model-based techniques for verification of properties of the composition (and not to build the composition itself). In general, works in this category provide support to the most basic workflow patterns (namely, sequence and AND-Split). One exception is the work presented in [29], which can generate service compositions with conditional constructor by using an interface automata model that supports non-deterministic choice.

Lastly, the Other Techniques category puts together works employing different techniques from those aforementioned. In the literature review conducted, works related with two others research areas were found: databases and evolutionary computing. The approaches employing databases techniques (more specifically, query rewriting) presented a expressiveness level similar to the others categories, i.e., giving support up to the parallel constructor. On the other hand, the work [61], which employs a technique from evolutionary computing (genetic programming), presents a higher expressiveness level by supporting eXclusive OR. This is done by defining a grammar used to represent the chromosomes (programs on a composition language), which contains the sequence, split, split-join and choice terminals. The running example presented in that paper, as well as the used benchmarks, does not show how the method exploits the generation of compositions with choice constructor. Therefore, its not clear how the approach can generate conditional compositions.

As it can be observed in Table 4, the last two columns are empty, i.e., none of the surveyed articles provide support to the OR-split -join and Iteration constructors. The Iteration constructor is admittedly one of the most complex constructors in programming languages (as well in the workflow languages). As stated in [65], plan generation containing iterative constructors cannot be carried out in a fully automatic way. The OR-split-join pattern presents a choice where multiple branches may be chosen. The implementation of this pattern requires the existence of an external controller, that verifies when all the triggered branches of the constructor have finish their execution. The surveyed proposals do not seem to need to support the pattern.

5 Conclusion

This work presented a literature review on the existing automatic web services composition approaches. This review allowed to ascertain that most of these approaches have an intermediate expressiveness level (taking as basis the basic workflow patterns considered in this paper), i.e., they widely support the sequential and parallel constructs. With regard to the more complex workflow patterns, namely eXclusive OR (XOR), OR and iteration, these

are partially (or not) supported by the surveyed works. In particular, the XOR pattern is supported by a small number of works [48, 38, 57, 54, 29, 61]. Even if these works take XOR pattern into account, the examples used in those works to illustrate their methods do not exhibit how the XOR pattern is supported (they exploit only sequence and AND-split patterns). No proposals supporting the generation of compositions containing the OR-split-join or iteration patterns were found.

The analysis conducted in this survey does not claim to be exhaustive. The expressiveness results observed in the surveyed works provide additional subsidies that might reinforce the aforementioned limitations of the techniques, specially those concerning to the AI planning. However, this observation is based on the informations collected during the literature review and it is not intended to be a definitive conclusion for the sharp expressiveness limits of the techniques.

Some surveyed works (like [66, 15, 37, 6, 13, 56, 52, 30, 67]) apply some kind of preprocessing phase during the composition process. In general, this preprocessing phase is concerned with calculating the semantic similarity between ontological concepts (e.g., as Causal Link Matrix [40]) used in the annotation of input/output parameters of the web services, aiming to avoid the execution overhead of this task during the composition process and thus increasing its performance.

ACKNOWLEDGEMENTS

The author would like to thank to CAPES/Brazil, UTFPR/Brazil and PEDECIBA/Uruguay by financial support.

References

- [1] Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972.
- [2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Golan, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*. IBM, 2003.
- [3] F. Baader. Description logic terminology. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 485–495. Cambridge University Press, 2003.

- [4] F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. A description logic based approach to reasoning about web services. In *Proceedings of the WWW 2005 Workshop on Web Service Semantics (WSS2005)*, Chiba City, Japan, 2005.
- [5] Mahmoud Barhamgi, Djamel Benslimane, and Brahim Medjahed. A query rewriting approach for web service composition. *IEEE Trans. Serv. Comput.*, 3:206–222, July 2010.
- [6] Peter Bartalos. Effective automatic dynamic semantic web service composition. *Information Sciences and Technologies Bulletin of the ACM Slovakia*, 3(1), 2011.
- [7] Peter Bartalos and Maria Bielikova. Automatic dynamic web service composition: A survey and problem formalization. *Computing and Informatics*, 30(4):793–827, 2011.
- [8] G. Baryannis and D. Plexousakis. Automated Web Service Composition: State of the Art and Research Challenges. *Technical Report ICS-FORTH/TR-409*, October 2010.
- [9] Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Trans. Program. Lang. Syst.*, 7(1):113–136, January 1985.
- [10] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic composition of e-Services that export their behavior. In *Proceedings of the International Joint Conference on Service Oriented Computing (ICSOC)*, pages 43–58, 2003.
- [11] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation, GECCO '05*, pages 1069–1075, New York, NY, USA, 2005. ACM.
- [12] K.S.M. Chan, J. Bishop, and L.: Baresi. Survey and comparison of planning techniques for web service composition. Technical report, Dept Computer Science, University of Pretoria, 2007.
- [13] Viorica R Chifu, Ioan Salomie, Agota Riger, and Valentin Radoi. A graph based backward chaining method for web service composition. *2009 IEEE 5th International Conference on Intelligent Computer Communication and Processing*, pages 237–244, 2009.
- [14] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Service Definition Language (WSDL). Technical report, March 2001.
- [15] Eduardo Gonçalves da Silva, Luís Ferreira Pires, and Marten van Sinderen. Towards runtime discovery, selection and composition of semantic services. *Comput. Commun.*, 34:159–168, February 2011.

- [16] Luca de Alfaro and Thomas A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, September 2001.
- [17] Jos De Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, Michael Kifer, Birgitta König-Ries, Jacek Kopecky, Rubén Lara, Holger Lausen, Eyal Oren, Axel Polleres, Dumitru Roman, James Scicluna, and Michael Stollberg. Web Service Modeling Ontology (WSMO). W3C Member Submission, June 2005.
- [18] A. E. Eiben and Marc Schoenauer. Evolutionary computing. *Inf. Process. Lett.*, 82(1):1–6, 2002.
- [19] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
- [20] Yuzhang Feng, Anitha Veeramani, Rajaraman Kanagasabai, and Seungmin Rho. Automatic service composition via model checking. *Asia-Pacific Conference on Services Computing.*, 0:477–482, 2011.
- [21] Matthias Fluegge, Ivo J. G. Santos, Neil Paiva Tizzo, and Edmundo R. M. Madeira. Challenges and techniques on the road to dynamically compose web services. In *Proceedings of the 6th international conference on Web engineering*, ICWE '06, pages 40–47, New York, NY, USA, 2006. ACM.
- [22] Malik Ghallab, Craig K. Isi, Scott Penberthy, David E. Smith, Ying Sun, and Daniel Weld. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
- [23] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, Amsterdam, 2004.
- [24] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
- [25] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, December 1995.
- [26] N. Guarino. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1st edition, 1998.
- [27] Giancarlo Guizzardi. *Ontological foundations for structural conceptual models*. PhD thesis, Enschede, 2005.
- [28] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, December 2001.

- [29] Seyyed Vahid Hashemian and Farhad Mavaddat. A graph-based approach to web services composition. In *Proceedings of the The 2005 Symposium on Applications and the Internet*, SAINT '05, pages 183–189, Washington, DC, USA, 2005. IEEE Computer Society.
- [30] Ourania Hatzi, Dimitris Vrakas, Mara Nikolaidou, Nick Bassiliades, Dimosthenis Anagnostopoulos, and Ioannis Vlahavas. An integrated approach to automated semantic web service composition through planning. *IEEE Transactions on Services Computing*, 5(3):319–332, 2012.
- [31] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978.
- [32] Jörg Hoffmann and Bernhard Nebel. The ff planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.*, 14(1):253–302, May 2001.
- [33] Hopcroft and Ullman. *Introduction to Automata Theory Languages and Computation*. Addison-Wesley Publishing Company, 1979.
- [34] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosz, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium, 2004.
- [35] John A. Kalman. *Automated Reasoning with OTTER*. Rinton Press, 2001.
- [36] Esra Kirci Ozorhan, Esat Kaan Kuban, and Nihan Kesim Cicekli. Automated composition of web services with the abductive event calculus. *Inf. Sci.*, 180(19):3589–3613, October 2010.
- [37] Matthias Klusch and Andreas Gerber. Semantic web service composition planning with owls-xplan. In *In Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, pages 55–62, 2005.
- [38] Srividya Kona, Ajay Bansal, M. Brian Blake, and Gopal Gupta. Generalized semantics-based service composition. In *In Proc. IEEE Intl. Conf. Web Services (ICWS)*, pages 219–227, 2008.
- [39] John R. Koza. *Genetic programming - on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press, 1993.
- [40] Freddy Lecue and Alain Leger. A formal model for semantic web service composition. In *IN ISWC 2006*, pages 385–398, 2006.

- [41] Daewook Lee, Joonho Kwon, Sangjun Lee, Seog Park, and Bonghee Hong. Scalable and efficient web services composition based on a relational database. *Journal of Systems and Software*, 84(12):2139 – 2155, 2011.
- [42] Fen Lin, Lirong Qiu, He Huang, Qing Yu, and Zhongzhi Shi. Description logic based composition of web services. In Zhong-Zhi Shi and Ramakoti Sadananda, editors, *Agent Computing and Multi-Agent Systems, 9th Pacific Rim International Workshop on Multi-Agents, PRIMA 2006, Guilin, China, August 7-8, 2006, Proceedings*, volume 4088 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2006.
- [43] Huan Liu, Farong Zhong, Bang Ouyang, and Jiajie Wu. An approach for qos-aware web service composition based on improved genetic algorithm. In *Proceedings of the 2010 International Conference on Web Information Systems and Mining - Volume 01, WISM '10*, pages 123–128, Washington, DC, USA, 2010. IEEE Computer Society.
- [44] Zohar Manna and Richard Waldinger. Fundamentals of deductive program synthesis. *IEEE Trans. Softw. Eng.*, 18(8):674–704, August 1992.
- [45] David Martin, Mark Burstein, Erry Hobbs, Ora Lassila, Drew Mcdermott, Sheila Mcilraith, Srini Narayanan, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic Markup for Web Services. Technical report, November 2004.
- [46] Amin Mesmoudi, Michael Mrissa, and Mohand-Said Hacid. Combining configuration and query rewriting for Web service composition. Technical Report RR-LIRIS-2010-015, LIRIS UMR 5205 CNRS/INSA de Lyon/Universite Claude Bernard Lyon 1/Universite Lumiere Lyon 2/Ecole Centrale de Lyon, July 2010.
- [47] Harald Meyer and Dominik Kuropka. Requirements for automated service composition. In *Proceedings of the 2006 international conference on Business Process Management Workshops, BPM'06*, pages 447–458, Berlin, Heidelberg, 2006. Springer-Verlag.
- [48] Harald Meyer and Mathias Weske. Automated service composition using heuristic search. In Schahram Dustdar, Jose Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2006.
- [49] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- [50] Robin Milner. The polyadic pi-calculus (abstract). In Rance Cleaveland, editor, *CONCUR*, volume 630 of *Lecture Notes in Computer Science*, page 1. Springer, 1992.

- [51] Tadao Murata and Du Zhang. A predicate-transition net model for parallel interpretation of logic programs. *IEEE Trans. Softw. Eng.*, 14(4):481–497, April 1988.
- [52] Cagla Okutan and Nihan Kesim Cicekli. A monolithic approach to automated composition of semantic web services with the event calculus. *Knowledge-Based Systems*, 23(5):440 – 454, 2010.
- [53] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 333–347, London, UK, UK, 2002. Springer-Verlag.
- [54] Petros Papapanagiotou and Jacques Fleuriot. Formal verification of web services composition using linear logic and the pi-calculus. In *Proceedings of the 2011 IEEE Ninth European Conference on Web Services*, ECOWS '11, pages 31–38, Washington, DC, USA, 2011. IEEE Computer Society.
- [55] Joachim Peer. Web Service Composition as AI Planning - a Survey. Technical report, University of St. Gallen, Switzerland, 2005.
- [56] C. V. S. Prazeres, C. A. C. Teixeira, and M. D. C. Pimentel. Semantic Web Services discovery and composition: paths along workflows. *Ecows'09: Proceedings of the 7th Ieee European Conference on Web Services*, pages 211+, 2009.
- [57] Jinghai Rao, Peep Kungas, and Mihhail Matskin. Composition of semantic web services using linear logic theorem proving. *Inf. Syst.*, 31(4):340–360, June 2006.
- [58] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Proceedings of the First international conference on Semantic Web Services and Web Process Composition*, SWSWPC'04, pages 43–54, Berlin, Heidelberg, 2005. Springer-Verlag.
- [59] Domenico Redavid, Luigi Iannone, Terry Payne, and Giovanni Semeraro. Owl-s atomic services composition with swrl rules. In *Proceedings of the 17th international conference on Foundations of intelligent systems*, ISMIS'08, pages 605–611, Berlin, Heidelberg, 2008. Springer-Verlag.
- [60] Pablo Rodriguez-Mier, Manuel Mucientes, and Manuel Lama. Automatic web service composition with a heuristic-based search algorithm. *Web Services, IEEE International Conference on*, 0:81–88, 2011.
- [61] Pablo Rodriguez-Mier, Manuel Mucientes, Manuel Lama, and Miguel I. Couto. Composition of web services through genetic programming. *Evolutionary Intelligence*, 3(3-4):171–186, 2010.

- [62] Murray Shanahan. Artificial intelligence today. In Michael J. Wooldridge and Manuela Veloso, editors, *Artificial intelligence today*, chapter The event calculus explained, pages 409–430. Springer-Verlag, Berlin, Heidelberg, 1999.
- [63] Murray Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, 44:207–239, 2000.
- [64] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. Htn planning for web service composition using shop2. *Web Semant.*, 1(4):377–396, October 2004.
- [65] Werner Stephan and Susanne Biundo. Deduction-based refinement planning. In *PROCEEDINGS OF THE 3RD INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE PLANNING SYSTEMS (AIPS-96)*, pages 213–220. AAAI Press, 1996.
- [66] Hassina Nacer Talantikite, Djamil Aissani, and Nacer Boudjlida. Semantic annotations for web services discovery and composition. *Computer Standards & Interfaces*, 31(6):1108 – 1117, 2009.
- [67] Xianfei Tang, Changjun Jiang, and Mengchu Zhou. Automatic web service composition based on horn clauses and petri nets. *Expert Syst. Appl.*, 38:13024–13031, September 2011.
- [68] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
- [69] Yves Vanrompay, Peter Rigole, and Yolande Berbers. Genetic algorithm-based optimization of service composition and deployment. In *Proceedings of the 3rd international workshop on Services integration in pervasive environments*, SIPE '08, pages 13–18, New York, NY, USA, 2008. ACM.
- [70] Bochao Wang, Armin Haller, and Florian Rosenberg. Generating workflow models from owl-s service descriptions with a partial-order plan construction. *Web Services, IEEE International Conference on*, 0:714–715, 2011.
- [71] Yan Xu, Bin Li, and Jun Wu. A petri-net coverability model for automatic web service composition. In *Proceedings of the 2009 International Conference on Industrial and Information Systems*, IIS '09, pages 31–34, Washington, DC, USA, 2009. IEEE Computer Society.
- [72] Li Ye and Junliang Chen. Automatic composition of semantic web services; a theorem proof approach. In *Proceedings of the First Asian conference on The Semantic Web*, ASWC'06, pages 481–487, Berlin, Heidelberg, 2006. Springer-Verlag.