A graph-based reduction in PlanICS abstract planning, based on partial orders of services (Extended Abstract)

Maciej Szreter

Institute of Computer Science, Polish Academy of Sciences

Abstract. The paper deals with the abstract planning problem – a stage of the Web Service Composition in the PlanICS framework. The planning task is viewed from the graph perspective, searching a graph built of vertices representing service and object types, and edges connecting service types to object types processed by each corresponding service. The preprocessing search identifies all the service and object types, which can potentially participate in a plan bounded by the given length. Then, the planning problem is split into subproblems, at the basis of the relation of independence between the sets of object types listed in user query as the desired result of the composition. The ontology is divided into disjoint sub-ontologies, each of which contains only the types relevant for the respective set. If there is a sub-plan for every sub-ontology, the resulting plan is composed out of these sub-plans.
The presented approach uses similarily defined graphs, and makes use of planners as external tools, as in [Szr15] describing graph-based pruning of ontologies, however the aim is different. In [Szr15], there are removed all the service and object types which, for the given user query, cannot occur in any plan bound by a fixed length. The current work optimizes planning in what is left by this reduction.

1 Introduction

The key concept of Service-Oriented Architecture (SOA) [Erl05] consists in using independent (software) components available via well-defined interfaces. Frequently, there is no web service directly satisfying the user objective, but a composition of services can deliver the requested result. An automatic composition of Web services should relieve the user of a manual preparation of detailed execution plans, matching services to each other, and of choosing optimal providers for all the components. The problem of finding such a composition is hard and well known as the Web Service Composition Problem (WSCP) [Erl05]. There is a number of various approaches to solve WSCP [LOKX10].

Automated composition of web services Web services are widely used to implement SOA paradigm, but much of their benefits is revealed when they can be composed automatically. The existing solutions to WSCP are divided into several groups. Following [LOKX10] our approach belongs to AI planning methods, including also approaches based on: automata theory, situation calculus, Petri nets, automated theorem proving, and model checking.

In the paper [SPAS03] Sycara et.al. present DAML-S, the predecessor of OWL-S, which is one of the standards to describe Semantic Web services. The authors show how to compose services using the DAML-S virtual machine and matchmaking mechanism with a support of a planning-capable agent. Another approach to WSC using OWL-S is described in [KG05] where Klusch et.al. introduce the OWLS-Xplan framework.

Graph-based approaches to planning and web services composition Several papers use graph-based approach to the WSCP. An example is [HM06], where information about inputs and outputs of services is represented by interface automata. The key difference between these papers and our approach is that we model matching of the service input and output at the graph level, by reducing it to the problem of testing reachability in graphs. Graph vertices model not only services, but also objects read and produced by services. Due to introducing the abstract planning we can reduce matching of service and object types to the problem of existing paths between selected graph vertices. The other approaches use graphs as a model for representing the state space, but test matching objects to services by calling specialized algorithms. This can hamper the performance. Another original feature is using graph databases for representing graphs and doing operations on them. To the best of our knowledge, we could also find no experimental analysis showing that graph-based approaches can effectively deal with ontologies with significant numbers of service and object types.

Graph databases [RWE13] are a relatively recent addition in the domain of the NoSQL databases, defined by rejecting the traditional database model of relational tables, and using alternative solutions, in this case graphs. Neo4j [Lal15] is one of the most popular implementations.

2 Solution

For the efficiency reason, the planning process in PlanICSis divided into two phases: abstract and concrete planning. The former deals with matching types, and the latter with matching the exact values of attributes. The current paper deals only with the former one. Here, we briefly recall the formulation of the abstract planning problem (APP) in PlanICS. We present only the intuitions, but all the formal definitions can be found in [D$^+$11].

PlanICS has much in common with well known approaches to composition of web services in the Semantic Web environment. One of similarities between PlanICS and DAML-S/OWL-S is the description of service capabilities. That is, both the approaches use the implicit capability representation, i.e., a service is described by the state transformation, its input, and output. This paradigm is often called IOPE - Input, Output, Precondition, Effect. Services process objects: read those placed in their input list, and write or produce those in output lists.

In addition, objects from inout lists can be both read and written. We refer to all these lists as *input and output lists*. Objects have as attributes either primitive types such as integers, booleans, and strings, or other objects. Pre- and post-conditions formulae, assigned to service types, determine the state of (some of) these attributes before and after executing the service. All the service and object types are organized in an inheritance hierarchy and represented in an *ontology*. A *user query* represents the aim of the composition, with similar input and output lists, and pre-/postcondition formulae as for services, determining the initial and the expected state of the composition process. The precisely defined semantics defines which sequences of services are the solution of the user query.

Semantics of abstract planning In abstract planning, the aim is to find the abstract plan(s). A *world* is a set of objects with valuations assigned to their attributes. An *transformation* is an application of a service to an *input* world, producing an *output* world such that the valuations of objects from both worlds occuring in the lists for the service are consistent with the pre- and postcondition formulae. A *solution* is a sequence of service types, with a *context function* assigning instances of objects to the input and output list of every service. A solution explains which transformations are needed in order to deliver all the object types requested in the user query, with the appropriate attributes set or not. Note that the objects needed by services can be provided in the input lists of the user query, or by services producing objects while taking no input. Two solutions are *equivalent* if they contain the same number of occurences for every service type. An *abstract plan* is an equivalence class with respect to this relation, so that the irrelevant orderings of services are abstracted away.

Currently there are three PlanICS solvers, based on SMT [NP13], Genetic Algorithms (GA) [SNP13] and the hybrid (H) [NPS15] being the combination of SMT and GA. For testing the performance, an ontology generator has been implemented, with several parameters characterizing the randomly generated ontologies. These parameters include the minimal and maximal numbers of services, objects, object attributes, objects in service lists, and objects in the user query. The structure of the plans is determined by the number of partial orders out of which which every object from the final world can be constructed.

The performance of the planners exhibits some general characteristics. SMT planner can handle smaller ontologies than GA, but finds all the plans and can determine that no plan exists at all. The hybrid planner combines the features of both SMT and GA planner. A common feature of all the planners is that their performance degrades when the length of the solutions increases. Also, all the plans found for the given depth need to be of the equal length.

Graph-based reduction of ontology In [Szr15] we have shown a graph-based pruning approach removing from an ontology all the service and object types which are not relevant for any plan of the given length, as determined by the user query. The search is implemented using a graph database. This improved significantly the performance of all the tested planners, and it corresponds to a typical scenario where only a small part of the ontology is relevant for the user query. Now we briefly describe this reduction.

First, the *ontology graph* is constructed for the given ontology, so that every service type and every object type are uniquely represented by a graph vertex. For every vertex representing a service type, there are incoming edges from vertices for the object types present in the input lists of the service, and there are outgoing edges to vertices for the object types in the output lists.

For the given user query, the additional two vertices are added to the ontology graph: the start and final vertex. The start vertex has outgoing edges to every object type from the input lists of the user query. The final vertex has incoming edges from every object type from the output lists of the user query. Then, we search for all the paths leading from start to final vertex, yielding *query k-subgraph* G_{qs}. We proved that only the service and object types present in G_{qs}, with their supertypes and the subtypes of the object types from the query, can occur in any plan of the length bound by k, and all the other types can be pruned.

Some pruned ontologies are still hard for the planners. We diagnosed that the problem is caused by the length of the plans rather than by their number. Usually, even restricting the ontology with more plans only to the types occuring in a single plan does not improve the situation.

Paper contribution: finding sub-ontologies in the reduced ontology Now we will focus on more efficient planning for the ontologies pruned by the graph reduction. In particular, we identify the disjoint sub-ontologies which can be checked independently, so that the resulting plans can be composed by taking a sub-plan found for every sub-ontology. The sub-ontologies are identified by using the graph approach based on analysis of G_{qs}. In particular, for every object type occurring in the output lists of the user query, we define a *Object Type Derivation Graph* (OTDG) which is G_{qs} restricted to the paths going via the vertex representing this type.

Then, we introduce the independence relation between the subsets of object types occurring in the output lists of the user query. Two such subsets are independent iff for every pair of object types from both these sets, their OTDGs are disjoint (have no common vertices). For every subset, the *sub-ontology* contains all the service and object types present in the OTDGs for the object types from the set, and the predecessors of these types. The user query is restricted to sub-queries accordingly.

The correctness of the reduction is shown in the following way. We claim that the set of plans generated by our construction at the basis of sub-plans found for every sub-ontology is equal to the set of plans for the k-reduced ontology. To show this, first we prove that every plan composed of sub-plans for sub-ontologies is a plan in the k-reduced ontology. Then, we show that for every plan from the k-reduced ontology, there exists a plan composed from the elements of union of a sub-plan for every sub-ontology such that these two plans are equivalent.

Note that if there is no sub-plan for a sub-ontology, then no plan exists for the complete ontology.

Experiments We briefly describe experiments in which we compared the performance of two planners (SMT and GA) for three approaches: the full ontology (FO), the k-reduced ontology (RO) and the sub-ontologies (SO). The examine two groups of benchmarks produced by a scalable generator accepting several parameters. There are five objects of distinct object types to be produced ($n_{|EW|} = 5$). 102 object types and 64 service types are present in every ontology. In the group A the length for every subplan is $len = k/n_{|EW|}$, compared to k being the length of every plan for FO and RO. There are 4 plans, because two object types can be produced in two ways, and other object types in a single way. In the group B, we scale the number of ways in which some two object types can be produced. The length of every plan is thus constant but the number of plans changes.

We implemented the described algorithm generating the k-query subgraphs for every query, and determining the sub-ontologies corresponding to the subgraphs determined by our independence relation. Its running time is very short (well below 1 second) for all the benchmarks presented in the paper.

A	$len = 2 + id_1, n_{\|EW\|} = 5, k = 5 * len, \|p\| = 4$											
id_1	SMT_{FO}		SMT_{RO}		SMT_{SO}		GA_{FO}		GA_{RO}		GA_{SO}	
	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t
1	4	7.7	4	3.2	4	0.4	4	3.9	4	2.7	4	1.3
2	4	88.5	4	34.8	4	0.7	4	6.4	3	4.4	4	1.3
3	4	1025	4	560	4	1.1	2	12.1	3	6.9	4	1.4
4	4	TO	4	TO	4	2.3	0	20.6	1	11.9	4	1.5
5	4	TO	4	TO	4	2.9	0	32.7	0	19.9	4	1.6
6	3	TO	3	TO	4	3.2	0	50.7	0	30.2	4	1.8
7	0	TO	1	TO	4	3.9	0	73.9	0	49.3	4	2.1

B	$len = 2, k = 10, n_{\|EW\|} = 5, \|p\| = 2^{id_2}$											
id_2	SMT_{FO}		SMT_{RO}		SMT_{SO}		GA_{FO}		GA_{RO}		GA_{SO}	
	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t	$\|p\|$	t
1	2	1326.5	2	549.9	2	1.7	2	7.0	2	4.8	2	1.6
2	4	785.8	4	384.0	4	1.7	4	7.1	4	4.7	4	1.8
3	8	1610.8	8	861.5	8	1.8	5	6.6	6	5.2	8	1.9
4	16	TO	16	TO	16	2.0	6	6.7	7	7.5	16	2.1
5	32	TO	32	TO	32	2.1	6	10.4	6	5.3	32	2.3

Table 1: Experimental results for two sets of parameters. $|p|$ is the number of plans, t time in seconds. TO denotes times longer than 2000 seconds. For GA, 10 instances have been run, and we report the maximal number of plans found.

The conclusion is that the sub-plans are found very quickly, because they are much shorter than the complete plan.

3 Conclusion

As the experiments testify, the presented algorithm can shorten the planning times by several orders of magnitude, when it is applicable. Conceptually, it is different from the well-known partial-order reductions used in formal verification. The key difference is that the latter approaches deal with global states, possibly pruning some executions not relevant for preserving the requested properties. Here we identify independent sub-systems at the basis of their local behavior.

The future research will consist in proposing weaker independence conditions, relaxing the requirement for disjoint OTDGs. Another research direction is to encode OTDGs directly into a planner. This would enable finding plans of different lengths for a fixed depth bound. An ultimate aim is a fully graph-based planner directly exploating the structure of the problem.

References

[D$^+$11] Dariusz Doliwa et al. PlanICS - a web service compositon toolset. *Fundam. Inform.*, 112(1):47–71, 2011.

[Erl05] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[HM06] Seyyed Vahid Hashemian and Farhad Mavaddat. A graph-based framework for composition of stateless web services. In *ECOWS*, pages 75–86. IEEE Computer Society, 2006.

[KG05] Matthias Klusch and Andreas Gerber. Semantic web service composition planning with owls-xplan. In *Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, pages 55–62, 2005.

[Lal15] Mahesh Lal. *Neo4J Graph Data Modeling.* Packt Publishing, 2015.

[LOKX10] Zheng Li, Liam O'Brien, Jacky Keung, and Xiwei Xu. Effort-oriented classification matrix of web service composition. In *Proc. of the Fifth International Conference on Internet and Web Applications and Services*, pages 357–362, 2010.

[NP13] Artur Niewiadomski and Wojciech Penczek. Towards SMT-based Abstract Planning in PlanICS Ontology. In *Proc. of KEOD 2013 International Conference on Knowledge Engineering and Ontology Development*, pages 123–131, September 2013.

[NPS15] Artur Niewiadomski, Wojciech Penczek, and Jaroslaw Skaruz. Hybrid approach to abstract planning of web services. In *Service Computation 2015 : The Seventh International Conferences on Advanced Service Computing*, pages 35–40, 2015.

[RWE13] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases.* O'Reilly Media, Inc., 2013.

[SNP13] Jaroslaw Skaruz, Artur Niewiadomski, and Wojciech Penczek. Automated abstract planning with use of genetic algorithms. In *GECCO (Companion)*, pages 129–130, 2013.

[SPAS03] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1:27–46, 2003.

[Szr15] Maciej Szreter. Bounded abstract planning in PlanICS based on graph databases. Technical Report 1031, ICS PAS, Ordona 21, 01-237 Warsaw, December 2015.