PlanICS 2.0 - a web service composition system

Abstract

Distributed web services with well-defined interfaces enable building complex functionalities from simpler ones. An automatic web service composition prepares an execution plan specifying how to reach a given goal, fitting the services together and choosing an optimal provider for each required service type. PlanICS 2.0 is a web service composition system implementing our original approach aimed at providing flexibility at the level of modelling the reality in which the web services operate, and enabling to handle the services that do not publish their internal semantics, but communicate only by simple query/answer entries. PlanICS 2.0 separates between an abstract and a concrete planning phase, where the former deals with service types while the latter with their concrete instances, thus making the matching more efficient. Another distinguishing feature of the system consists in defining a computation engine as an independent block, which enables to compute plans using any suitable approach. Currently, two engines, based on a genetic algorithm and an SMT-solver, have been implemented. The paper presents PlanICS 2.0 at a general level, comparing it also to related solutions from the area of automated web service composition.

Keywords: automatic web services composition, IOPR model, semantic web, ontology reasoning

1 The research described in this paper has been supported by the National Science Centre under the grant No. 2011/01/B/ST6/01477.
1. Introduction

Automatic composition of web services [2, 1, 12] is a relatively fresh research area, gaining momentum as a web service-based infrastructure is becoming more and more popular. The problems to be solved are of very broad scope: syntactic matching of different description languages and approaches, dealing with semantic differences, high complexity associated with a large number of distributed services, various formulations of goals to be reached, etc. PlanICS 2.0 is a system implementing our original approach which solves the composition problem in some clearly separated stages. Fig.1 shows the general PlanICS 2.0 architecture. The information about the services is stored in the following way: an ontology, managed by the ontology provider, contains a system of classes describing the types of the services as well as the types of the objects they process, while the service registry keeps an evidence of real-world web services, registered accordingly to the service type system. PlanICS 2.0 uses a state-based approach, which means that there are states (worlds) representing (partial) 'snapshots' of the reality, and services transforming them by modifying object attributes and adding new objects. Composition is thus understood as searching for a set of services capable to process certain states in a desired way.

![Diagram of PlanICS 2.0 system architecture](image)

**Figure 1.** A diagram of PlanICS 2.0 system architecture. The bold arrows correspond to computation of a plan, the thin arrows model the planner infrastructure, the dotted arrows represent the user interaction.
The user expresses a goal by a query, referring to objects and adding constraints, and defining an initial world to start with and an expected world to be reached. The system searches for a service composition transforming a subset of the initial world into a superset of the expected world. The latter, obtained by executing services according to a plan, is called a final world.

The composition process looks as follows: in its first stage, an abstract planner produces a (context) abstract plan, matching services at the level of input/output types. In the second stage, this plan is used by an offer collector, i.e., a tool which queries real-world services. The result is an offer plan containing concrete offers produced by service instances of appropriate types. In the third stage, the offers are searched by a concrete planner in order to find the best solution maximising a quality function.

PlanICS 2.0 has been revised and extended comparing to its previous edition [4, 5], making it easier to adapt to real-world applications (a comparison of the two versions is provided in the final section). This paper gives an overview of PlanICS 2.0 in a way strict but informal, because of the space limitations. The rest of the paper is structured as follows. In Section 2 the basic notions are introduced, necessary to describe the key topic of planning in Section 3.

1.1. Related work

The research in the area of automatic web service composition started very briefly after web services themselves became an important part of the modern IT. Many different approaches have been put forward, with several aims, ideas, and solutions. Here, we briefly describe the state-of-the-art in the field.

The Entish system [1] and the WSMO/SESA project [15] are two approaches particularly close to PlanICS 2.0 by sharing the idea of using ontologies with a formal semantics for representing knowledge about services. The major common features shared with Entish are discovering service capabilities by web communication, multi-stage planning, and using a similar service description language (a restricted quantification has been introduced to PlanICS 2.0). WSMO is similar to PlanICS 2.0 with respect to expressing the goal as a
system state and using mediators/proxies for communicating with real-world services. PlanICS 2.0 differs from both the systems by a service model, extending the IOPE descriptions with mapping inputs/outputs of services to states of the transformed worlds, and an automatic conversion of the planning problem to the abstract domain (for planning in service types). Another difference is that our system does not (yet) execute services.

Among other approaches to service composition, [11] tackles the problem as a logic-based program synthesis using theorem provers. A semi-automatic composition is described in [13], and special languages to describe plans were proposed in [6] and [10]. An important group of methods formulates service compositions in terms of AI-planning. One of the most commonly used planning approaches is STRIPS/PDDL, used for example in [9].

Testing of the developed solutions is sometimes problematic as there are still not sufficiently many available real-world services. Thus, own testbeds were developed, enabling composition testing while setting the parameters of services [3]. PlanICS 2.0 also implements such a tester [7, 14].

2. Basic notions

Below we introduce basic notions for describing the planning stages of PlanICS 2.0.

2.1. Objects, object types, ontology

One of the main assumptions of our approach is that all the web services in the domain of interest as well as the objects processed by the services can be strictly classified in a hierarchy of classes, organised in an ontology (the ontologies are encoded using the OWL language [8]). All the classes are derived from the base class Thing. There are three direct descendants of Thing, namely Artifact, Service, and Stamp. The rest of the ontology modelling the domain of interest can be designed in an arbitrary way, but not violating the rules presented below.

The branch of classes rooted at Artifact is composed of the types of objects the services operate on, while the branch rooted at Stamp contains types
of special-purpose objects aimed at confirming service executions and describing certain execution features (like a price or an execution time). Each object type definition consists of a number of typed attributes specifications, with the set of types including integer and real numbers, boolean values, dates and references to other objects. An object of a given type is an instance of the appropriate class.

The rules of class inheritance: a subtype class contains all the attributes of its parent classes, and optionally introduce some more. Multi-base inheritance is also allowed. The names of the attributes are unique within the ontology.

Valuations of objects, worlds: An object valuation is a function that assigns to each attribute of the object a value from the respective domain. A world is a set of objects together with their valuations. If partial valuations for a set of objects are specified only, then they define a set of worlds, of elements determined by all the possible assignments for the missing values, covering the respective domains. By a sub-world of a world \( w \) we mean a restriction of \( w \) to some subset of objects from \( w \). Given two objects \( o, o' \) and their valuations \( v_o, v_{o'} \) we say that \( v_{o'} \) is compatible with \( v_o \) if the type of \( o' \) is either the same as the type of \( o \) or is a subtype of that type, and the values of \( v_o \) and \( v_{o'} \) are the same for all the common attributes of the objects.

Consequently, a world \( w \) is compatible with a world \( w' \) if there exists a one-to-one mapping between the objects of \( w \) and \( w' \) such that each object from \( w' \) is compatible with the object of \( w \) it corresponds to.

### 2.2. Services

A key notion of the approach is that of a service. We assume that each service processes a set of objects, possibly changing values of their attributes, and produces a set of new (additional) objects. The types of services available for planning are defined as elements of the branch of classes rooted at Service. Each service type stands for a description of a set of real-world services of some common features.

The common features of the services of a given type are described using the attributes introduced by the Service class. These attributes are: \( \text{in}, \text{inout}, \text{out} \),
and out aimed respectively at specifying sets of objects the service of a given type requires to execute (leaving them unmodified), processes while the execution (possibly modifying) and produces as its result, preCondition and postCondition (pre and post, for short) aimed at specifying the conditions these objects are to satisfy, and inquiry, offer, and assign enabling to describe the interaction with a real-world service of this type (an intuition behind the service description is presented in Fig. 2). Technically, the values of pre and post are Boolean formulas (encoded in strings, similarly as the other features of services; an interpretation of these strings is the main task of the PlanICS 2.0 parser) being combinations of expressions over attributes of the objects from in, inout and out (respecting type limitations) and functions from a certain set applied to these attributes. In turn, the values of inquiry and offer are sets of (typed) parameters specifying respectively the data to be sent to a real-world service of the given type, and the data which will be received as an answer. The value of the assign attribute is a set of assignments specifying a relation between the contents of inquiry and offer and the attributes of objects from the sets in, inout and out. The values of all the above attributes are kept in the ontology as the valuation of a special instance of the corresponding class, called a metaservice.

**Figure 2.** PlanICS 2.0 service model. The boxes correspond to in, out, inout of a service, the puzzle-shapes model objects, the dots within them - their attributes.
The inheritance rules for the classes from the Service branch are the same as for the rest of the ontology. However, additional rules are needed to describe a computation of effective values of the attributes of metaservices being instances of derived classes. So, the formulas pre and post of such a metaservice are conjunctions of the corresponding formulas of all the ancestors up to the root of the hierarchy, and the formula specified explicitly. Similarly, the sets in, inout, out, inquiry, offer, and assign are unions of the appropriate sets for all the ancestors, and the set given explicitly. However, if the same object name is used in a set in a parent and a child class, then the one from the descendant must belong to a class derived from that from the ancestor, which means that it overrides the corresponding object from the parent specification.

**Semantics:** A service type \( s \) is understood as a pair of world sets, called the input and the output worlds, respectively. The input worlds consist of objects given by the union of the sets in and inout, of the valuations determined by the pre formula. Similarly, the output worlds are determined by the union of the sets in, inout, out, and the post formula.

A service type \( s \) can transform a world \( w \) if some its sub-world is compatible with some input world of \( s \). The result of such a transformation is a world \( w' \) composed of the set of objects obtained by enriching that of \( w \) by the objects produced by \( s \) (i.e., given by its out). The valuation of each object \( o \in w \cup w' \) not used as the inout parameter of \( s \) is the same in \( w \) and in \( w' \), the valuations of all the remaining objects from \( w \cap w' \) (i.e., used as the inout parameters of \( s \)) can by different from these in \( w \) only if this is implied by the assign attribute of \( s \), and the objects from \( w' \setminus w \) (produced by \( s \)) have their valuations assigned in a way resulting from the assign attribute of \( s \). The valuations satisfy also the post formula of \( s \). By a transformation sequence we mean a sequence of service types such that the first service type is able to transform a given world, and each subsequent service type is able to transform the result of the previous transformation.

**Service registry:** The service registry is an element of the system which keeps an evidence of real-world web services, registered by their providers accordingly to the service types given by the ontology. Each entry of the registry corresponds to one real-world service (however, the service provider can
register its functionality using a number of PlanICS 2.0 service registry entries, for example to declare its compatibility to several PlanICS 2.0 services), and is a tuple containing an unique identifier of the service (assigned by the system), a type of the service (taken from the ontology), its specific \textit{pre} and \textit{post} that express conditions to be satisfied by the data the service receives and returns (the conditions are therefore logical formulas over the components of \textit{inquiry} and \textit{offer} for the appropriate class), and an \textit{offerBinding} program responsible for interacting with the real-world service and obtaining this way offers satisfying requirements of interest.

3. Planning

Planning is the core functionality of PlanICS 2.0. In this section, we describe the complete planning process, starting from the user query and going through all the planning stages.

3.1. A user query

A task the user expects from the system to perform is given in the form of a user query specification. It resembles a service definition, i.e., contains typed objects in the \textit{in}, \textit{inout}, and \textit{out} sets, as well as \textit{pre} and \textit{post} formulas over their attributes. As in the case of the service types, an interpretation of a user query specification is a pair of world sets. The initial worlds are determined by objects from \textit{in} and \textit{out}, and the \textit{pre} formula of the query, while the expected worlds are defined by its \textit{in}, \textit{inout}, \textit{out}, and the \textit{post} formula.

A user query specification can introduce additional constraints on the world obtained as a result of composition (the final world) in order to limit the number of objects of a particular type (\textit{cardinality constraint}) or aggregated (by aggregating we mean taking sum, minimal or maximal value) values of certain object attributes (\textit{aggregate constraint}). Moreover, a \textit{quality function} enables to specify criteria for evaluating the quality of a plan (e.g., the minimal cost, the minimal time, or some more complicated expression over objects from the final world).
Thus, every transformation sequence able to transform some initial world into a final world, which contains some expected world and meets all the additional constraints, and satisfies the user query, is called a user query solution.

### 3.2. Abstract planning

The first stage of planning is performed by an abstract planner. Its main goal is to determine which service types can potentially cooperate to satisfy the user query, and thereby to reduce the number of interactions with services in the subsequent planning phases. The planning in this phase looks as follows. First, the pre and post formulas from the service types and the user query specifications are converted to abstract formulas. That is, according to a formally defined transformation, the expressions involving the object attributes are simplified and each attribute value is substituted with the predicate isSet or isNull. These predicates determine the cases when the value needs to be known or remains unassigned, respectively. The transformation considers also the relations between reference attributes of objects and (in part) cardinality restrictions specified in the user query. Then, a search is performed and a context abstract plan (CAP) is produced. It specifies which service types need to be applied over which objects, in order to satisfy the user query. By the contexts we mean mappings between objects in the worlds, and the objects being service parameters.

### 3.3. Collecting offers

The next stage of the process consists in collecting offers. This is done by an offer collector on the basis of the context abstract plan, but using full (not abstract) condition formulas.

The offer collector communicates with the registered real-world web services of appropriate types (using their offerBindings to this aim), collecting offers for each service type present in the context abstract plan. More precisely, the offer collector sends to each appropriate offerBinding the constraints on the data we are potentially able to sent to the service in the inquiry, and on the
data we expect to receive in the offer in order to keep on building a potential plan (checking earlier whether these constraints do not contradict the specific limitations of the service specified in its pre and post in the registry). The offerBinding program determines, by way of an interaction with the real-world service, the possible variants of the service execution satisfying the constraints mentioned, and returns them in the form of formulas. The pair of formulas, the first of which specifies the actual constraints on the data the real-world service "agrees" to receive, and the second - the constraints on the data it declares to return in consequence, is called a proposal of the service.

The offer collector works recursively, using the proposals collected earlier to obtain the next ones, and memorising the bindings between them. The result of its work is an offer plan of the nodes representing sets of worlds implied by the proposals and mapped to the worlds of CAP, and of the edges corresponding to real-world services of appropriate types.

### 3.4. Concrete planning

The last stage of the planning process is concrete planning, taking an offer plan and finding a concrete plan - i.e., a sequence of real-world services (corresponding to a CAP sequence) and data to be sent to these services which form together a scenario maximising the quality function. Potential consecutive phases of planning, such as executing concrete plans or regenerating them partially when the execution fails, are currently not covered by our research.

### 3.5. Implementation, algorithms

At the current stage of the PlanICS project, abstract planners have been implemented with the associated infrastructure. Concerning ontology, the OWL modeling approach is used with available implementations. Two abstract planners have been developed so far: one based on a translation to Satisfiability Modulo Theorems (SMT) [7] and another using Genetic Algorithms approach [14]. A generator of benchmarks has been implemented, allowing to scale several parameters such as number of services, maximum number of processed objects and object attributes, etc.
Conclusions and future work

PlanICS 2.0 offers a complete solution to automatic web service composition, distinguished by a multi-stage planning and focused on an easy adaptation to existing models of web services. Our main objective was to create a reasoning system based on ontologies modelling selected aspects of business processes, implemented as web services. A flexible semantic model allows to use the system in various domains in order to achieve a goal - automatic web service composition. Additional advantages of the approach are a reduction of the search space by a classification of the services, and dynamic discovering their capabilities suitable for the created composition. A special attention is paid to transforming a composition problem to other problems solvable by well-known effective reasoning methods.

Comparing PlanICS 2.0 to its previous edition [4, 5], the improvements consist in: (1) extending the service descriptions by parts related to proxy communication (inquiry, offer) and relation of the processed world with the service input and output (assign), (2) introducing a concise multi-proposal representation of offerBinding, (3) separating between collecting offers and planning with offers, (4) extending the query language with a restricted quantification over objects and attributes as well as with a quality measure, (5) defining a formal conversion from the concrete to the abstract planning domain, and (6) providing two implementations of the abstract planning engines based on genetic algorithms and SMT-solvers. The offer collecting and concrete planning phases are still to be implemented as a next stage of the development.

Bibliography


