A Distributed multi-agent planning approach for automated web services composition

Mohamad El Falou, Maroua Bouzid, Abdel-Illah Mouaddib, Thierry Vidal

A distributed multi-agent planning approach for automated web services composition

Mohamad El Falou [a,*], Maroua Bouzid [a], Abdel-Illah Mouaddib [a] and Thierry Vidal [b]
[a] *GREYC Laboratory, University of Caen, Basse-Normandie, France*
E-mail: {melfalou,bouzid,mouaddib}@info.unicaen.fr
[b] *LGP Laboratory, Ecole Nationale d'Ingenieurs de Tarbes, France*
E-mail: thierry.vidal@enit.fr

Abstract. The ability to automatically answer a request that requires the composition of a set of web services has received much interest in the last decade, as it supports B2B applications. It aims at selecting and inter-connecting services provided by different partners in response to client requests. Planning techniques are used widely in the literature to describe web services composition problem. However, since web services proliferate day after day, classical planners are no longer well suited to compose web services in a reasonable time. This weakness is due to the explosion of the search space caused by the large number of services and the broad range of data exchanged among services. Therefore it is more interesting to use a decentralized planner to distribute the search space and the computing load taking into account the distributed nature of the problem. In this paper, we propose a distributed multi-agent approach to solving the web services composition problem at runtime. Our approach consists of a set of web services agents where each agent has a set of services organised in a graph. To respond to a request, agents propose their best local partial plans which are partial paths in the graph. They then coordinate their partial plans to provide the best global plan for the submitted request. The analysis of the complexity and the results of the implementation show the ability of our approach to scaling up when compared to the state-of-the-art techniques for automated web services composition.

Keywords: Web services, multi-agent, distributed planning

1. Introduction

Web services are distributed software components that can be exposed and invoked over the Internet using standard protocols. They communicate with their clients and with other web services by sending XML based messages over the Internet [13]. They are self-contained, self described, active and modular software applications that can be advertised, discovered and invoked over the Internet [14]. For example, an airline travel service or a book-buying service (amazon.com).

Many languages have been developed in industrial and academic communities to specify web services, such as BPEL4WS (Business Process Execution Language for Web Services) [3], WSDL (Web Services Description Language) [12].

*Corresponding author.

Research in the area of web services is increasingly focusing on the problem of the automated composition of web services: given a set of services that are published on the web, and given a goal, the aim is to generate a composition from the available services that satisfies the goal.

Artificial Intelligence planning techniques can be used to model the composition problem. In fact, services can be modeled as actions and the business process as a plan to connect the web services. The system builds the plan by sensing the world to explore the available services. It develops from the initial state the different intermediate states, by applying services, until reaching the goal state. It returns a plan which is a set of services allowing to reach the goal, or it returns failure.

Since the number of services available over the web increases dramatically, centralized approaches are not

efficient for this kind of problem because both the number of services available over the web, and the number of possible combinations of services, increase dramatically, leading to a state explosion in the planning search space.

A few studies have been dedicated to develop decentralised techniques. In this paper we propose a distributed planning algorithm for web services composition where each agent is defined by a set of services. It builds, from the set of services, a graph describing their relationships. When an agent receives a request for composing web services, it first computes the best partial plan. Then it coordinates with other agents by merging partial plans in order to find a global plan that responds to the submitted request. Each agent computes its best partial plan, using a heuristic function.

First, the heuristic is defined by a local function which computes the distance between the local state and the goal state. The empirical results and the study of the complexity show the effectiveness of the decentralised approach to composing web services at runtime. On the contrary, this approach has a limitation. It is not complete when the agents are Dependant.

Second, the heuristic is extended by a global distribution which ensures the completeness of the approach by avoiding the wells and considering the intermediate local plans offered by other agents when computing the best local plan.

The remainder of this paper is organised as follow: Section 2 presents the centralised and decentralised approaches to compose web services. In Section 3, we give a motivated example. The formal framework is developed in Section 4. In Section 5, we explain our multi-agent approach based on local heuristic which is extended in Section 6 finally, this paper is concluded in Section 8.

2. Related work

In this section we present some recent works on the composition of web services in the centralised and decentralised planning frameworks.

2.1. Centralised approaches

The work on web service composition presented in [1] describes web services as 3 abstract processes. Given n 3 abstract processes, the system will automatically translates each of them into a state transition system (STS) $D = \langle S, A, I, T \rangle$ where S is the set of state, A is the set of actions, $I \subset S$ is the set of initial states, and $T : S \times A \rightarrow 2^S$ is the transition function. After translating web services, the system constructs a parallel product $\sum_{\|}$ which combines the n STS; this parallel product allows the n services to evolve concurrently. From $\sum_{\|}$, they generate a planning domain that is passed as an input to the planner. The planning problem consists in finding a plan π defined by a set of actions allowing to reach a defined goal and satisfying some constraints on the execution of the plan. They use the Model Based Planner MBP [5] based on model checking techniques [4]. The drawback of this approach is the recalculation of $\sum_{\|}$ whenever a service is added or removed from the domain.

In his dissertation [7], Alexander Lazovik defines the problem of service composition in the same manner used above. However, he adds properties allowing interaction with the client during the execution phase based on the principle of interleaving the planning and the execution phases. His planner is based on constraint satisfaction techniques; he motivates his work by the fact that in a web service scenario, users may wish to know why certain solutions are preferred over others.

Because users' needs and interests are in a constant evolution, a conversation-driven composition of web services is developed in [31]. It ensures that the composition of web services efficiently handles these changes by dynamically allowing to choose with whom they would like to interact. Web services would be able to engage conversations, make decisions and adjust their behavior according to the situations in which they participate. Even though that work addresses dynamic composition of web services, it does not consider goal-driven planning capabilities, but rather focuses on adjusting the composition of evolving web services through conversations, while we are more interested here in the way a composition is built from scratch, starting from the user's request.

In [10] a system called GOLOG and based on the situation calculus is presented. It composes web services by applying logical inference techniques on predefined plan template. Given a goal description, the logic programming language GOLOG is used to instantiate the appropriate plan for composing the services described in Prolog. GOLOG is implemented in Prolog and based on the situation calculus. It supports the specification and the execution of complex actions in dynamic systems.

In [18], the authors define a translation from DAML-S process models to the SHOP2 domains, and from the DAML-S composition tasks to a SHOP2 planning problem. SHOP2 is a well-suited planner for working with the process model in a Hierarchical Task Network (HTN). HTN planning builds plans through task decomposition.

Grid techniques are used in [22] to automatically generate workflows by: reducing the search space, selecting appropriate services to apply and improving the planner performance and the quality of plans in a centralised architecture. Two planners were used in this approach: PRODIGY [23] based on learning techniques and FF [24] which uses heuristics that estimate the distance to the goal.

In [25] and [19], Freddy Lécué composes the web services by focusing on the functional level of web services. Web services composition is then viewed as a composition of semantic links controlled by causal laws. The semantic links refer to semantic matchmaking between web services parameters (output and input) in order to model their connection and interaction whereas causal laws are the relationships between actions, action preconditions and side-effects. Then, two different approaches are developed. The first is based on the semantic links matrix which is required as a starting point to apply problem-solving techniques such as regression-based search. In the second approach, in addition to semantic links, causal laws are also considered to achieve composition. To this end an augmented and adapted version of the logic programming language Golog is used.

This work has been extended in [32] for automatically composing non-deterministic services. In this context, extended Golog operates as an offline interpreter that supports n-ary sensing services (i.e. services conditioned on their possible output parameters) to achieve conditional composition of services.

Finally, in [21], two centralised algorithms based on tree-search and graphplan are developed to compose web services. In this approach, a new type of request defined by an initial and a goal state, is described. It allows to claim the creation and elimination of objects in any state.

According to the specification language used to define the web services, a planning technique is more preferred to compose web services. The main limitation of all centralised approaches cited above is their ineffectiveness to scale up well due to the state explosion in the planning search space.

2.2. *Decentralised approaches*

CASCOM [26] stands for Context-Aware business application Service Coordination in Mobile computing environments. Its main objective is to implement, test and validate an infrastructure for the business applications based on the semantic web through fixed and mobile networks. A multi-agent system is used for the selection, ordering and execution of the services.

In [28], Fiorino et al. define a planner agent for each service. The agent role is to simulate the performance of the service. This agent is an autonomous entity which contains a planner and is able to communicate with other agents in order to co-construct a plan for composing web services. A planner agent is based on the dialectical theory for plan synthesis. This approach devises a system based on planning agents in which the production of a global shared plan is obtained by conjecture/refutation cycles. The dialogue between agents is a joint investigation process allowing agents to progressively prune objections, solve conjectures and elaborate solutions step by step until that goal is reached or no further refinement is possible.

Yasmine Charif [27] proposes in her PhD thesis a multi-agent coordination model for dynamic service choreography. In this coordination model, the services collaboration capabilities are modeled through introspective agents capable of reasoning on their own services, to dynamically take part into a task achievement and to coordinate with other agents in a way that overcomes their limitations and covers the user needs.

To the best of our knowledge, there is no other multi-agent approach to compose web services. The main difference between those methods is: in the former CASCOM architecture, there is only one planning agent and the other agents are involved in other tasks (selection, execution and monitoring of services) to help the planning agent. In the latter approaches, however, a planning agent is associated with each web services. Therefore the load is shared among several agents. In this paper, we define a new methodology based on graph-based distributed heuristic search, to automatically compose web services. As opposed to other decentralised methodologies, it responds to the implicit web services request as it was introduced in [21], and it allows much more composition work offline, which improves the scalability of the overall composition system.

3. Motivating example

Let us consider a set of web services which are intended to deal with files, images and tracks as follows:

1. Web service WS_1 translates file languages. It has three services: $fr2en$ translates files from french to english, $en2ar$ translates files from english to arabic and $en2fr$ translates files from english to french.
2. Web service WS_2 transforms text file formats. It has two services: $latex2doc$ transforms files from latex to doc format and $doc2pdf$ transforms files from doc to pdf format.
3. Web service WS_3 merges files. It has two services: $mergePdf$ merges two pdf files into a third one and $mergedoc$ merges two doc files into a third one.
4. Web service WS_4 transforms image file formats. It has two services: $png2jif$ transforms an image from png to jif format and $jpeg2png$ transforms an image from jpeg to png format.
5. Web service WS_5 transforms the type of a track. It has two services: $wav2mp3$ transforms a track from wav to mp3 format and $rm2mp3$ transforms a track from rm to mp3 format.

As an example, let us suppose that we have two files: the first is in doc format written in English, the second is in latex format written in French and we want to obtain a file which contains the content of the two files translated to Arabic.

4. Formal framework

Our formal framework is based on extended Planning-Graph techniques [16] allowing the creation and elimination of objects when executing services (actions).

Contrary to classical approaches where a state is defined as a set of predicates, a state in our domain is defined by a set of objects, properties and relations between these objects. The definition of actions is also extended to allow the generation and elimination of objects in the environment, the assignment of new predicates to objects and the definition of new relations between them.

4.1. Basic idea of the multi-agent approach

The web services composition domain is defined by a set of web services: $\langle WS_1, \ldots, WS_n \rangle$. A planning agent $\mathcal{A}_i$ is associated with each WS_i respresented by a graph of services. A Central Agent $\mathcal{A}_c$ plays the role of an interface between a client request $R = (init, goal)$ and the composition system, and manages a portion of the coordination between the web services agents. When a client sends to the system a request R, the central agent $\mathcal{A}_c$ broadcasts R to all the agents $\mathcal{A}_i$.

Each agent computes its best local plan to reach $goal$ from $init$. The central agent sorts all partial plans based on their heuristics then merges them in order to obtain a global plan Π_1. Π_1 is applied on $init$ to obtain $init_1$. Then, it builds a new request $R_1 = (init_1, goal)$ and broadcasts it to all agents. The steps cited above are called a planning phase. Planning phases are repeated within a forward search strategy (see Fig. 1) until reaching a phase in which $init_i$ equals $goal$ and a solution is found, or $distance(init_i, goal) > distance(init_{i-1}, goal)$ which is the failure case and no solution is found. In the next section, our formal framework is defined. It uses a part of the definition of the composition problem defined in [21] and extends it to distributed settings.

4.2. Preliminaries and definitions

The domain $D = (C, P)$ is defined by a set of web services $C = (WS_1, WS_2, \ldots, WS_n)$ that we call a community of web services, and a set of predicate types $P = \{p_1, p_2, \ldots, p_n\}$ to specify the possible properties of objects and relations between them.

In the motivating example (Section 3), the community of web services is defined by C= (WS_1, WS_2, WS_3, WS_4, WS_5) and the set of predicate types is defined by $P = \{(en\ F), (merge\ F1\ F2\ F), (jpeg\ I), (mp3\ T), \ldots\}$ where F, I and T are respectively a file, an image, and a track. The predicate $(en\ F)$ is a property of F which means that F is in English, $(merge\ F1\ F2\ F)$ is a relation which means that F is a merge of $F1$ and $F2$, etc.

Definition 1. A state $q = (O, P)$ of the plan execution is defined by a set of objects O and their types, and a set of predicates P specifying the properties of these objects and the relationship between them.

In Section 3, the initial state is specified as: $init = [\{(F1 : file), (F2 : file)\}, \{(doc\ F1), (en\ F1), (latex\ F2), (fr\ F2)\}]$, where $F1$, $F2$ are objects (files) and file is a type and doc, en, latex, and fr are properties.

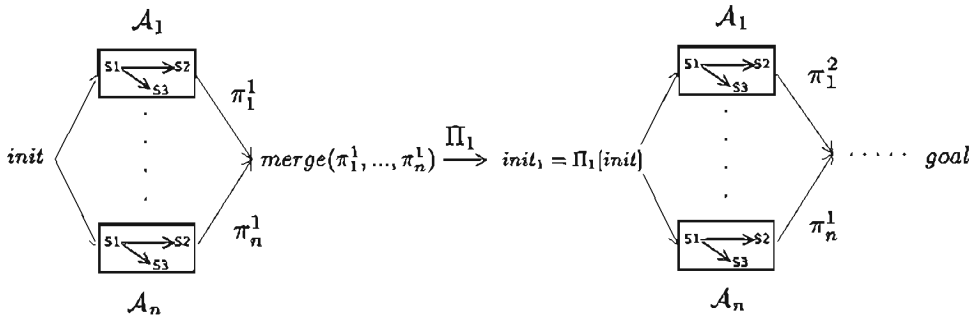

Fig. 1. Forward strategy.

Definition 2. A web service WS is defined by $WS = (T, I, S)$ such that: T is the type of service, I is the set of web service attributes, $S = (s_1, \ldots, s_m)$ is the set of services available within WS.

In the example, $WS_1 = (translation,\ IP = \text{'127.127.0.41'}, (fr2en, en2ar, en2fr))$.

Definition 3. A service s in a web service WS is defined by $s = (Pin, Pout, Pinout, Prec, Effect)$ where:

- $Pin = \{pin_1, \ldots\}$ are input objects of s;
- $Pout = \{pout_1, \ldots\}$ are output objects of s;
- $Pinout = \{pinout_1, \ldots\}$ are input-output objects of s;
- $Prec$ is the set of conditions to be satisfied by objects of Pin;
- *Effect* is the set of *explicit* execution effects of the service in the current state of the domain.

We notice that we actually have two kinds of effects for the execution of a service; implicit and explicit ones. The implicit effects, namely eliminate all the input variables in Pin and create new variables for each output parameter in $Pout$[1]. The explicit effects consist of deleting the predicates of $Prec$ and adding the predicates of *Effect*.

In the example *mergePdf* of WS_3, the service is defined as follows:

- $Pin = \{(F1 : file), (F2 : file)\}$.
- $Pout = \{(\#F : file)\}$.
- $Pinout = \{\}$.
- $Prec = \{(pdf\ F1), (pdf\ F2)\}$.
- $Effect = \{(pdf\ \#F), (merge\ F1\ F2\ \#F)\}$.

[1]The execution of a service leads to the reservation of its inout objects *Pinout*, but with the possibility to change their predicates and their relations.

There is a difference between the STRIPS [30] representation of the action and our representation of the service. In fact, STRIPS representation distinguishes between the preconditions $Prec$ and the deleted Del predicates. After an action execution, only the set of predicates in Del are deleted from the world state. This is due to the fact that the STRIPS representation is generally used in physical domains, and not all the preconditions must be deleted after an action execution. For example, in robotics domains, to put an object on a table, the robot must be next to the table and still next to the table after putting the objects.

In our model, since we model web services which deal with objects to change their properties or the relations between them, we simplify our representation of service. So we don't define the set of deleted predicates Del and we suppose that all the preconditions are deleted when executing a service. Our model can be easily extended to a STRIPS model. For example, to represent that a precondition predicates must not be selected as effect of an action execution, one must simply add this predicate to the effect set as in the example of the $en2fr$ service represented in Section 5.6.

An agent graph of services G is defined for each web service.

Definition 4. A graph of services $G = (S, E)$ consists of a set of services S and a set of edges E to link an output parameter (or input-output) of one service s_i to an input (or input-output) parameter of another service s_j. An edge has a direction represented by an arrow link, from the output of a service to the input of another service.

An edge a_{ij}^{mn} connecting two services s_i and s_j is defined between an output parameter out_i^m of s_i and an input parameter in_j^n of s_j if and only if:

- $type(out_i^m) = type(in_j^n)$.
- $prec(in_j^n) \subseteq effect(out_i^m)$.

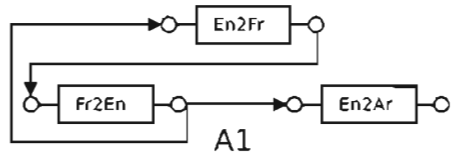

Fig. 2. Agent graph of services for WS_1.

The graph of services of WS_1 in Section 3 is sketched in Fig. 2.

The $fr2en$ service is defined as follows:

- $Pin = \{\}$.
- $Pout = \{\}$.
- $Pinout = \{(F1 : file)\}$.
- $Prec = \{(fr\ F1)\}$.
- $Effect = \{(en\ F1)\}$.

$en2ar$ can be defined in the same way.

An edge is defined between the output parameter out of $fr2en$ and the input parameter in of $en2ar$, because:

- $type_{fr2en}(out) = type_{en2ar}(in) = File$,
- $prec_{en2ar}(out) \subseteq effect_{fr2en}(in) = (en)$.

The edge can be viewed as a link to represent a link of type 'producer/consumer'. $fr2en$ produces an english file which is consumed by $en2ar$.

Definition 5. A plan is defined by a sequence of sets of services. More formally $\Pi = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$ is a plan such that $\forall i \in [1 \ldots n], \pi_i = (s_i^1, \ldots, s_i^n)$ is a set of independent services, and each s_{ik} is instantiated with real objects in the domain.

The execution of the plan $\Pi = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$ consists of executing in order the set of services $\pi_i, i \in [1 \ldots n]$. However, there are no order constraints related to the execution of services in π_i since we can execute them in any order, even in parallel.

One plan solution for the problem introduced in Section 3 is $\Pi = \langle \pi_1, \pi_2, \pi_3, \pi_4, \pi_5 \rangle$ where:

- $\pi_1 = (en2ar[F1], fr2en[F2])$;
- $\pi_2 = (doc2pdf[F1], en2ar[F2])$;
- $\pi_3 = (latex2doc[F2])$;
- $\pi_4 = (doc2pdf[F2])$;
- $\pi_5 = ([\#0file] = mergePdf[F1, F2])$.

Definition 6. A service s is *executable* in a state q if and only if:

- For every object $o \in \{P_{in} \bigcup P_{inout}\}$ there exists a variable in the current state q which is of the same type as o.
- The set of preconditions on input and inout objects are all satisfied.

In Section 3, $doc2pdf[F1]$, $en2ar[F1]$, $latex2doc[F2]$, $en2fr[F1]$, and $fr2en[F1]$ are executable services in the $init$ state.

After the execution of a service, the state of the world changes. New variables are generated (the output parameters of the executed service), and new predicates are applied to the set of the generated variables. Further updates will be made for all objects of P_{inout} based on the effects of the service.

Definition 7. A request $R = (init, goal)$ is defined by the initial state $init$ and the goal state $goal$.

In the previous example, $init = [\{(F1 : file), (F2 : file)\}, \{(doc\ F1), (en\ F1), (lat\ F2), (fr\ F2)\}]$ and $goal = [\{(\#F0 : file)\}, \{(pdf\#F0), (ar\ F1), (ar\ F2), (merge\ F1\ F2\ \#F0)\}]$.

The objects of which identifiers start by # consist of generated objects. The aim of using the symbol # before the name of the object is to state that it is a generated object (in the output set of the executed service), and any other object having the same type beginning with # can replace it in the domain. We can hence distinguish such generated objects from the initial objects which are defined in the initial state of the request (they do not start with the # symbol).

Definition 8. Let $q_1 = (O_1, P_1)$ and $q_2 = (O_2, P_2)$ be two states, $q_1 = q_2$ if and only if:

- $O_1 = O_2$ apart from renaming generating objects;
- $P_1 = P_2$ after satisfying the first condition, and using the same renaming in the set of predicates.

Let:

- $q_1 = [\{(\#F0 : file), (F1 : file)\}, \{(en\ \#F0), (ar\ F1)\}]$;
- $q_2 = [\{(\#F : file), (F1 : file)\}, \{(en\ \#F), (ar\ F1)\}]$;
- $q_3 = [\{(\#F0 : file), (F : file)\}, \{(en\ \#F0), (ar\ F)\}]$

be three states, we have $q_1 = q_2$, but $q_1 \neq q_3$ nor $q_2 \neq q_3$.

In fact, $q_1 = q_2$ because the generated object $\#F0$ can replace $\#F$ because they have the same type, but F cannot replace $F1$ because they are not generated objects.

Definition 9. The distance between two states $q_1 = (O_1, P_1)$ and $q_2 = (O_2, P_2)$ is defined as follows: $distance(q_1, q_2) = (|O_1| + |O_2| - 2 * |O_1 \cap O_2|) + (|P_1| + |P_2| - 2 * |P_1 \cap P_2|)$ when neither O_1 nor O_2 contain generated objects. If at least one contains generated objects then $distance(q_1, q_2)$ equals the minimum of the distances between all possible renaming objects.

Example Let us remind the *init* and the *goal* states of our motivated example 3 which are defined as follows:

- $init = [\{(F1 : file), (F2 : file)\}, \{(doc\ F1), (en\ F1), (lat\ F2), (fr\ F2)\}]$;
- $goal = [\{(\#F0 : file)\}, \{(pdf\ \#F0), (ar\ F1), (ar\ F2), (merge\ F1\ F2\ \#F0)\}]$.

So, $distance(init, goal) = (2 + 1 - 2 * 0) + (4 + 4 - 2 * 0) = 11$.

Definition 10. The heuristic function of a partial plan $h(\pi)$, applied in a state q for a request $R = (init, goal)$ is equal to $distance(q', goal)$, where $q' = \pi[q]$ is the state resulting from the application of the partial plan π in q.

Example In the motivating example, the heuristic of $\pi_1 = \langle\{en2ar[F1],\ fr2en[F2]\},\ \{en2ar[F2]\}\rangle$ is $h(\pi_1) = distance(\pi_1[init], goal) = (2 + 1 - 2 * 0) + (4 + 4 - 2 * 2) = 7$, where:

- $\pi_1[init] = init_1 = \{(F1 : file), (F2 : file)\}, \{(doc\ F1), (ar\ F1), (lat\ F2), (ar\ F2)\}$;
- $init = [\{(F1 : file), (F2 : file)\}, \{(doc\ F1), (en\ F1), (lat\ F2), (fr\ F2)\}]$;
- $goal = [\{(\#F0 : file)\}, \{(pdf\ \#F0), (ar\ F1), (ar\ F2), (merge\ F1\ F2\ \#F0)\}]$.

So, $distance(init, goal) = (2 + 1 - 2 * 0) + (4 + 4 - 2 * 0) = 11$.

5. Multi-agent approach using local heuristic

In this section, we explain our multi-agent approach based on local heuristic. First, we explain how to extract the best local plan from an agent graph (Section 5.1); second, how two agents collaborate by merging their best plans (Section 5.2). Then, different strategies for multi-agent coordination are presented (Section 5.3) and finally we discuss the empirical results.

5.1. Plan extraction

First of all, each agent computes off line the graph of services $GS = (S, E)$ which defines links between its services. Then, based on the local heuristic, the agent extract all reachable executable services from G in a given state *init*. Finally it computes the best local plan based on the distance to the *goal*.

We give here the basic idea of the algorithm to extract the sub-graph of all reachable executable services: first, a new colouring service is created s_c. This service will act as a catalyst to helping colouring all the reachable services from *init*. This service is defined by its output parameters, which are the objects of the *init* state, and its effects, which are the predicates of the *init* state.

Then, s_c is coloured and all its possible links with services are created. This colour is broadcasted to all services, for each neighbor service s non-coloured, s is coloured under the condition that all edges incoming to it leave a coloured service. This operation is repeated until no service is colorable. Finally, s_c is deleted. The subgraph GS_e defined by the set of coloured services S_e and edges between them E_e is extracted.

Example 1. Let us take a WS defined by six services illustrated in Fig. 3: Suppose that:

- $init = (\{(F1 : file), (F2 : file)\}, \{(doc\ F1), (pdf\ F2)\})$;
- $goal = \{(\#F0 : file)\}, \{(doc\ \#F), (merge\ F1\ F2\ \#F0)\}$.

The executable subgraph is illustrated in Fig. 4. It is obtained from the graph in Fig. 3 by adding as the first step the colouring service s_c. Services *pdf2doc* and *doc2pdf* are coloured because they have incoming arcs from s_c. Similarly, the service *mergePdf* is coloured because its two incoming arcs leave coloured services (s_c, *doc2pdf*). Services *txt2lat*, *lat2pdf* and *lat2eps* are not coloured because they are non-executable and unreachable from *init*.

5.1.1. Instantiation graph of executable sub-graph

The instantiation graph of services GI represents all the possible executable plans of the sub-graph of executable services $GS_e = (S_e, E_e)$ corresponding to a request $R = (init, goal)$. This graph is represented by a node-based tree as illustrated in Fig. 5. Each node represents a reachable state from *init* and each arrow represents an executable service, which, when it is executed on its left node, gives the right node as a result.

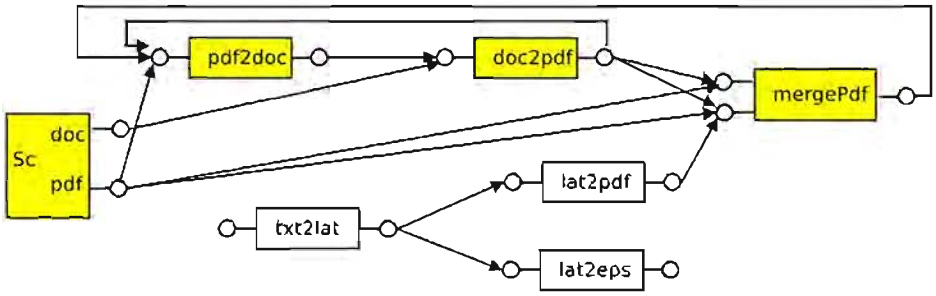

Fig. 3. Graph of services.

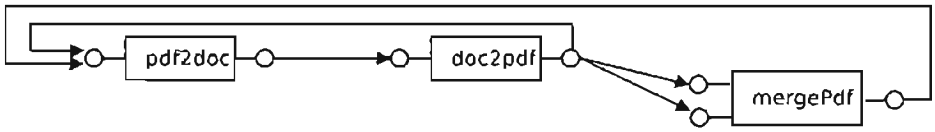

Fig. 4. Sub-graph of executable services.

Each path in the graph starting with the *init* state is a local agent plan.

The Algorithm 1 shows how to develop the instantiation graph from an executable graph GS_e. This graph is developed in a breadth-first search strategy. In line 2, S_i is initialised by all executable services on *init*. The vector V is initialised in line 3 by applying the services of S_i on *init* (States are inserted in order in the tail). Then, each non-developed state is developed until reaching the *goal* or all the nodes are developed (*while* loop in line 4).

For each iteration, state q is removed (line 5) from the head of the set of non developed states V. q is developed by applying all the successor services S' (loop *foreach* in line 8) of its predecessor service (s in line 6). In line 10, only the non-duplicated states in the path from *init* to q are added.

Example 2. Figure 5 illustrates the instantiated graph of the sub-graph of executable services illustrated in Fig. 4 developed for the request R of Example 1.

The execution of the Algorithm 1 is given below: in the initial state, services *doc2pdf* and *pdf2doc* are executed on the *init* state (line 3), so $V = [q1, q2]$. At line 5, the state $q1$ is removed from V to be developed. Its predecessor service is *doc2pdf*. The successor services of *doc2pdf* in Fig. 5 are $\{pdf2doc, mergePdf\}$.

There are two ways to execute *pdf2doc* on $q1$ (line 9) which are: $pdf2doc[F1]$, $pdf2doc[F2]$. $pdf2doc[F1]$ returns a state equal to *init*, so it is not added to V (line 10). $pdf2doc[F1]$ returns the state $q5$

Algorithm 1: Instantiated graph of executable sub-graph

```
1  Graph_instantiation(GS_e, init, goal)
2  S_i = executable(S_e, init)
3  V = expand(S_i, init)
4  while (¬stop) && (V ≠ ∅) do
5      q = V.remove()
6      s = predessor(q)
7      S' = successor(s)
8      foreach s' ∈ S' do
9          Q = expand(s', q)
10         V.add(Q)
11         if goal ∈ Q then stop = true
```

which is added to V. The same iteration is repeated until reaching the *goal* state $q6$.

5.1.2. Best partial plan extraction

The basic idea to extract the best local plan is to compute the distance between each state q_i in the instantiation graph and the goal. This set is then sorted to obtain the set of states $\langle q_1, \ldots, q_n \rangle$. Let $p_1, \ldots, p_n$ the set of associated partial paths to reach respectively $q_1, \ldots, q_n$. Hence the best partial plan is equal to $\pi = merge(p_1, merge(p_2, \ldots merge(p_{n-1}, p_n)))$.

The use of the *merge* operation locally allows to extract not only the best local path to reach the goal, but the set of all possible local paths which can be executed concurrently.

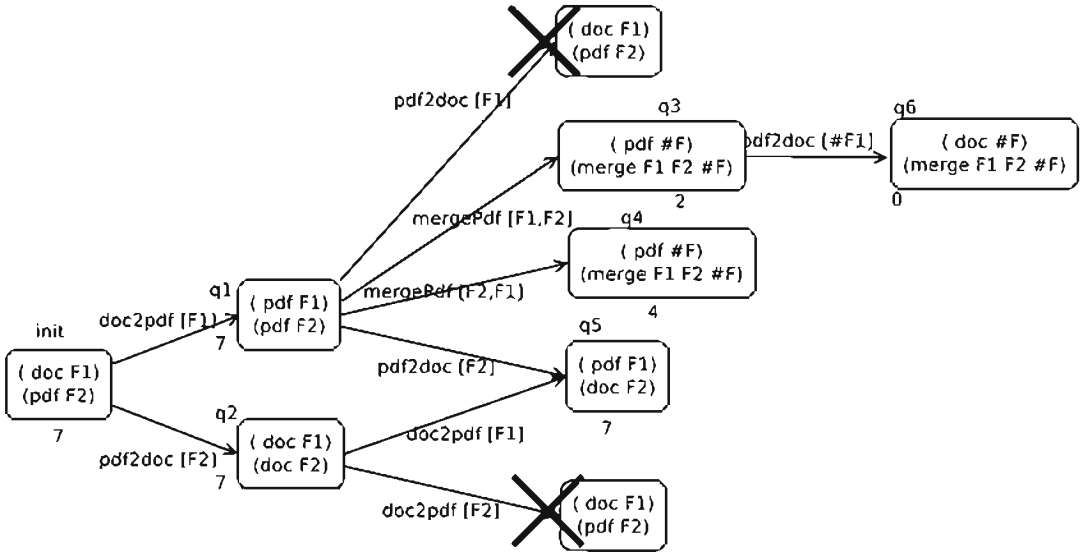

Fig. 5. Instantiated graph of the sub-graph of executable services.

In Fig. 5, $distance(q7, goal) = 0$. So the best partial plan is: $\langle doc2pdf[F1], \#f = mergePdf[F1, F2], pdf2doc[\#F1]\rangle$.

5.2. Plan merging operation

Collaboration between two agents $\mathcal{A}_1$ and $\mathcal{A}_2$ is achieved by merging the best plan $\Pi_1 = \langle \pi_1^1, \ldots, \pi_1^m \rangle$ of $\mathcal{A}_1$ with the best plan $\Pi_2 = \langle \pi_2^1, \ldots, \pi_2^n \rangle$ of $\mathcal{A}_2$ to obtain a third plan Π which is defined as follows:

- *Plans are completely mergeable* if and only if: $prec(\Pi_1) \cap prec(\Pi_2) = \emptyset$ and $objets(\Pi_1) \cap objets(\Pi_2) = \emptyset$ where $prec(\Pi_1)$, $prec(\Pi_2)$ are respectively the union of all pre-conditions for all services of Π_1, Π_2 and $objects(\Pi_1)$, $objects(\Pi_2)$ are respectively the union of all input objects for all services of Π_1, Π_2. In this case,
  - $\Pi = merge(\Pi_1, \Pi_2) = \langle \pi_1, \ldots, \pi_l \rangle$ where $l = max(m, n)$, m and n are respectively the size of Π_1 and Π_2.
  - $\pi_i = \pi_i^1 \cup \pi_i^2$ if $i < min(m, n)$.
  - $\pi_i = \pi_i^1$ if $m > n$ and $\pi_i = \pi_i^2$ if $n > m$.

  Example: let $\Pi_1 = \langle latex2doc[F1], doc2pdf[F1]\rangle$, and $\Pi_2 = \langle fr2en[F2], en2ar[F2]\rangle$, then the merged plan of Π_1 and Π_2 is $\Pi = \langle \{latex2doc[F1], fr2en[F2]\}, \{doc2pdf[F1], en2ar[F2]\}\rangle$.
- *Plans are partially mergeable* if and only if: $prec(\Pi_1) \cap prec(\Pi_2) = \emptyset$ and $objets(\Pi_1) \cap objets(\Pi_2) \neq \emptyset$.

  Note: we assume that Π_1 has a better heuristic cost than Π_2.
  This is the case when a plan contains $fr2en[F1]$ and $doc2pdf[F1]$. The preconditions of execution are therefore different, but since each service reserves objects during its execution, the two plans cannot run in parallel. They can run sequentially instead. In this case, we will insert the partial plan of Π_2 between two partial plans of Π_1.
  The merged plan Π is defined iteratively as follows:
  Initialisation $\Pi = \Pi_1$ for $i = 1 \ldots m$, if objects(π_i^1) $\cap$ objects(π_i^2) = $\emptyset$ then $\pi_i = \pi_i^1 \cup \pi_i^2$. else π_2 is inserted in Π between π_i and π_{i+1}.
- *Un-mergeable plans* if the plans are neither completely mergeable nor partially mergeable, then they cannot be merged and $\Pi = \Pi_1$ (best of the two conflicting plans).

5.3. Multi-agent coordination

There are multiple strategies to coordinate the web services agents. In this section, we propose three kinds of coordination: forward, backward and mixed strategies.

5.3.1. Forward coordination strategy

When a client requests the system with R, the central agent $\mathcal{A}_c$ broadcasts R to all the agents $\mathcal{A}_i$ (the left side of Fig. 6). Then the agents will compute the best local plan which, when applied on $init$, will reach the

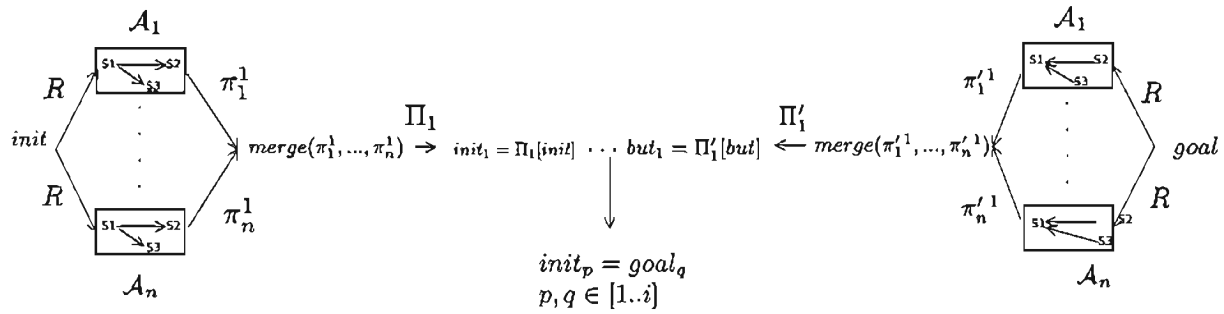

Fig. 6. Mixed strategy.

state the closest to *goal* (perhaps $\mathcal{A}_i$ doesn't have any applicable plan and the best plan is $\emptyset$).

Then, each agent sends its best local plan to the central agent $\mathcal{A}_c$. When $\mathcal{A}_c$ receives the responses from all agents, it sorts the plans by descending order $\langle \pi_1, \ldots, \pi_n \rangle$. Then it merges plans, one by one, to obtain a global plan, Π_1 is computed repeatedly by merging the partial plans in order. So, $\Pi_1 = merge(\pi_1, merge(\pi_2, \ldots merge(\pi_{n-1}, \pi_n)))$.

After that, $\mathcal{A}_c$ applies the merged plan Π_1 on the initial state $init$, it obtains a new state $init_1 = \Pi_1[init]$. Then by using $init_1$, $\mathcal{A}_c$ builds a new request which is defined by the new state $init_1$ and the goal state $goal$: hence $R_1 = (init_1, goal)$.

That process, called a *planning phase*, is then repeated: $\mathcal{A}_c$ distributes the request R_1 to all agents and so on until an agent (central or WS agent) reaches the goal state $init_i = goal$ and a solution is found, or $distance(init_i, goal) > distance(init_{i-1}, goal)$ which is the failure case and no solution is found.

5.3.2. Backward coordination strategy

We can use a similar sequence of planning phases, but using a backward strategy (right side of Fig. 6) instead of a forward strategy. The idea is to start from the goal and apply inverse services to produce subgoals, stopping if we produce a set of subgoals satisfied in the initial state.

An inverse service s^{-1} is defined for each service s. Effects of s become preconditions of s^{-1}, and the preconditions of s become the effects of s^{-1}. Input parameters become output parameters, and output parameters become input parameters.

Now, instead of each agent seeking the best local plan from the initial state to the goal state using its services, agents try to find a plan from the goal state to the initial state, using the inverse services. When a plan $\Pi = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$ is found from the *goal* to the *init* state, then the valid plan is its inverse which is equal to $\Pi^{-1} = \langle \pi_n, \ldots, \pi_2, \pi_1 \rangle$.

5.3.3. Mixed coordination strategy

Finally, forward and backward strategies can be applied together (Fig. 6). In this case, a planning phase consists in applying a forward strategy – for one time – on the initial state to obtain $init_1$, and in applying a backward strategy – for one time – on the goal state to obtain $goal_1$. After that, we obtain two new states $init_1$ and $goal_1$, we define a new request $R_1 = (init_1, goal_1)$. This request is then distributed to all agents until reaching two phases where $(init_i = goal_j)$ and a solution is found, or $distance(init_i, goal_j) > distance(init_{i-1}, goal_{j-1})$ which is the failure case and no solution is found.

Example of mixed strategy

Now we detail the behavior of the three agents $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ associated with the web services given in Section 3 and the behavior of the central agent. Each web services agent constructs links between its different services to construct the graph of services given in Fig. 7. The central agent $\mathcal{A}_c$ distributes the client request $R = (init, goal)$ to all the agents, where:

- $init = [\{(F1 : file), (F2 : file)\}, \{(doc\ F1), (en\ F1), (lat\ F2), (fr\ F2)\}]$;
- $goal = [\{(\#F0 : file)\}, \{(pdf\#F0), (ar\ F1), (ar\ F2), (merge\ F1\ F2\ \#F0)\}]$.

First iteration

Forward-strategy These are the responses of each agent when receiving the request R by applying services from $init$ to $goal$:

- $\mathcal{A}_1$: the best partial plan is $\pi_1 = \langle \{en2ar[F1], fr2en[F2]\}, \{en2ar[F2]\} \rangle$. Its heuristic value using Definition 10 is $h(\pi_1) = 7$.

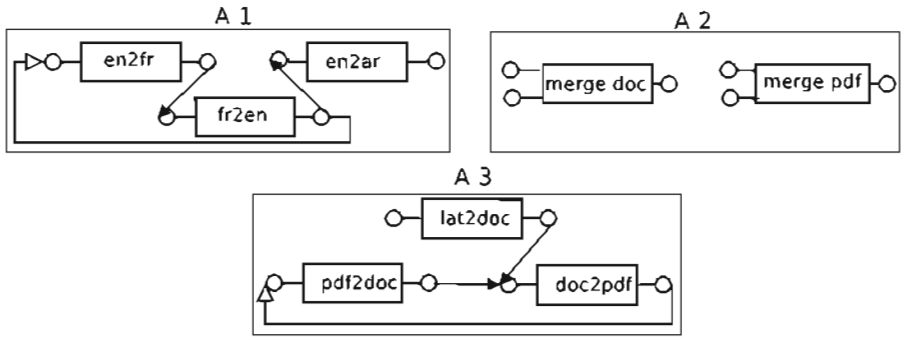

Fig. 7. Graphs of web services agents.

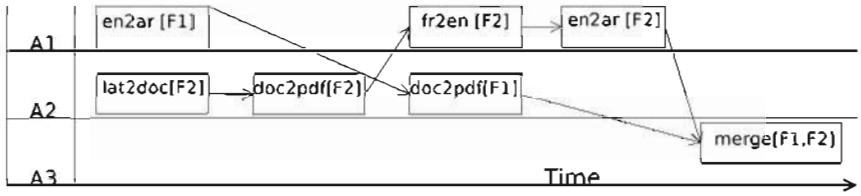

Fig. 8. Plan solution.

- $\mathcal{A}_2$: there is no executable plan, therefore $\pi_2 = \emptyset$. Its heuristic value is $h(\pi_2) = 11$ which is the distance between $init$ and $goal$.
- $\mathcal{A}_3$: the best partial plan is $\pi_3 = \langle\{doc2pdf[F1], lat2doc[F2]\}, \{doc2pdf[F2]\}\rangle$. Its heuristic is $h(\pi_3) = 7$.

Since both agents offer services with the same heuristic value, the list of partial plans is sorted as $\Pi = \langle\pi_1, \pi_3, \pi_2\rangle$. Thus, $\mathcal{A}_c$ computes $\Pi_1 = merge(\pi_1, \pi_3, \pi_2)$[2] which is equal to $\Pi_1 = \langle\{en2ar[F1], fr2en[F2]\}, \{doc2pdf[F1], lat2doc[F2]\}, \{en2ar[F2]\}, \{doc2pdf[F2]\}\rangle$.

Then, $\mathcal{A}_c$ computes $\Pi_1(init)$ and gets $init_1 = [\{(F1 : file), (F2 : file)\}, \{(pdf\ F1), (ar\ F1), (pdf\ F2), (ar\ F2)\}]$.

Backward-strategy These are the responses of each agent when receiving the request R by applying inverse services from $goal$ to $init$:

- $\mathcal{A}_1$: there is no executable plan, so $\pi'_1 = \emptyset$. Its heuristic is 11.
- $\mathcal{A}_2$: the best partial plan is $\pi'_2 = \langle mergePdf^{-1}[\#F]\rangle$ and $heuristic(\pi'_2) = distance(mergePdf^{-1}[\#F](goal), init)$ where $mergePdf^{-1}[\#F](goal) = \{(F1 : file), (F2 : file)\}, \{(pdf\ F1), (ar\ F1), (pdf\ F2), (ar\ F2)\}$. So $distance(mergePdf^{-1}[\#F](goal), init) = (2 + 2 - 2 * 2) + (4 + 4 - 2 * 0) = 8$.
- $\mathcal{A}_3$: there are two possible partial plans:
  - $p1 = \langle doc2pdf^{-1}[\#F]\rangle$;
  - $p2 = \langle doc2pdf^{-1}[\#F], lat2doc^{-1}[\#F]\rangle$.

  Hence $\mathcal{A}_3$ can propose any one of them, e.g. $\pi'_3 = \langle doc2pdf^{-1}[\#F]\rangle$ and $heuristic(\pi'_3) = 11$.

The list of partial plans is sorted by the central agent as $\langle \pi'_2, \pi'_3, \pi'_1\rangle$.

Thus, $\mathcal{A}_c$ computes $\Pi'_1 = merge(\pi'_2, \pi'_3, \pi'_1)$ which is equal to π'_2 because π'_3 is not mergeable with π'_2 and $\pi'_1 = \emptyset$.

Then, $\mathcal{A}_c$ computes $\Pi'_1(goal)$ and gets $goal_1 = [\{(F1 : file), (F2 : file)\}, \{(pdf\ F1), (ar\ F1), (pdf\ F2), (ar\ F2)\}]$.

At the end of the first iteration, $init_1 = goal_1$. So, our system finds the composition solution which is extracted by the central agent: the global plan is $\Pi = \langle\Pi_1, \Pi'^{-1}_1\rangle$ where $\Pi'^{-1}_1 = mergePdf[F1, F2]$. The plan description is given in Fig. 8.

5.4. Complexity

In this section, we compare the cost of using centralised and decentralised approaches to compose web services and we show the effectiveness of our decen-

[2]The order between partial plans is very important, as it may changes the results of the merge algorithm.

tralised approach to improving the scalability of the composition process. Our comparison is based on the complexity of each algorithm and on the number of messages exchanged among the agents in the decentralised model.

In [21], Tree-search and Graphplan algorithms are used to find the plan for web service composition. Graphplan is more effective than treesearch since [21]. The basic idea behind the Tree-search algorithm is to apply from the initial state all executable services. By doing this (expanding a state) we obtain a set of new states. Based on the strategy of Tree-search we select one of the unexpanded states to expand it. The expansion procedure is repeated until reaching the goal state and the solution is found or no state can be expanded which is the failure case. Tree-search complexity is $\mathcal{O}(b^p)$, where b is the branching factor of the graph and p is the depth of the solution. The Graphplan algorithm performs a procedure close to iterative deepening, discovering a new part of the search space at each iteration.

In contrast, in the decentralised architecture, computing the best partial plan is distributed among the agents and hence done in parallel, and a great part of computing is done offline by linking services. The search space is then also distributed among the agents. The complexity of a planning phase in the decentralised algorithm is the sum of:

- The complexity of finding the best partial plan by each agent which is $\mathcal{O}(m^2)$ where m is the maximum number of services of an agent A_i;
- The sorting operation computed by the central agent which is $\mathcal{O}(n * \log(n))$;
- The merging procedure applied to all the proposed partial plans which is $\mathcal{O}(n)$.

Let p the number of planning phases to found the solution, so the complexity of the decentralised algorithm is: $p * (\mathcal{O}(m^2) + \mathcal{O}(n * \log(n) + \mathcal{O}(n))$. This complexity is hence polynomial both in the number of agents and in the number of services per agent, for the three strategies.

The number of exchanged messages in the decentralised approach is not high. It equals $2n*(p)$, where n is the number of agents and p is the number of intermediate states $init1, init2, \ldots$. In fact, for each planning session, the central agent sends a message containing the query to each agent and receives as response a message containing the partial agent plan, so we have two messages per agent per planning session. In total, we have $2n * (p)$ message per n agents and p planning sessions.

5.5. *Implementation*

The first stage in planning is to define a language for the specification of the planning domain. In our implementation we use a part of the PDDL [17] language, which is intended to express the physics of a domain and extend it to fit our model.

In our architecture, we implement the central agent that contains a socket server to accept the connections of web services agents participating in our application. A web services agent is instantiated with the name of a file containing a set of services and the name of the web services composition server. Before sending a message to inform the server of its participation, an agent creates all the links between its services. When the server receives a request for the agent participation, it assigns to it a new thread (for sending queries and receiving partial plans). The server contains an interface to read the client request defined also in PDDL.

5.5.1. *Empirical results*

In our experimentation, we define eight web service agents. The number of services in each agent is in $[2, 8]$. The number of input parameters, output parameters, precondition predicates and effect predicates is in $[1, 3]$.

Agents are defined to deal with text, sound and picture files or goods. Services may convert sound tracks (rm to mp3, ...), convert picture formats (jif to jpeg, ...), convert file formats (pdf to doc, ...), translate text files (english to french, ...), merge tracks, documents and pictures files.

We tested our centralised and decentralised algorithms using the same heuristics (Definition 9) for a multiple set of queries. Results of four representative requests are given in Table 1. For each problem, we give respectively the number of objects in the query for the *init* and the *goal* states; the execution time of tree search (Tr) and of decentralised algorithms by using forward (Fo), backward (Ba) and mixed (Mi) strategies. In the last column the number of services in the solution plan are given.

Experiments show that when the query is simple, the centralised is faster than the decentralised approach, because of the time needed to communicate between agents.

In contrast, when the number of objects in the query becomes high, the composition problem becomes dif-

Table 1
Empirical results

Query	|Objects|		|Pred|		Execution time				Plan
	Init	Goal	Init	Goal	Tr	Fo	Ba	Mi	|Π|
P1	3	3	4	4	0.01	0.16	0.32	0.3	7
P2	3	1	6	6	0.02	0.57	0.66	0.62	7
P3	7	3	12	10	∞	0.69	6.14	4.03	13
P4	14	14	20	20	∞	0.68	3.23	1.48	18

∞ solution not found

ficult, and it requires the composition of a dozen services. Treesearch did not found any solution even after 15 minutes of execution (∞ symbol). But a decentralised algorithm (using any strategy) can find the solution after a few seconds as illustrated in P_3 and P_4.

Comparison between strategies shows that forward is faster than mixed which is faster than backward; the interpretation is that the cost of executing an inverse service s is greater than the cost of its normal execution.

Regression of a service is a bit more expensive than its normal execution especially when the service deletes objects. In fact, during the normal execution of a service creator of objects, any object name not previously used can be used to describe the created object (starts with character #). By cons, when executing an inverse service, the names of created objects should be selected among the names of objects of the same type of the *init* state.

Those experiments show the ability of our decentralised multi-agent approach to scaling up the applicability of state-of-the-art techniques for automated web services composition and confirm the theoretic complexity results.

5.5.2. *Comparison with the state-of-the-art*

To the best of our knowledge, the best recent results for the composition of web services are in Freddy Lécué's PhD thesis [25]. He presents three different scenarios:

- A Telecommunication scenario which contains 35 services;
- An e-tourism scenario which contains 45 services;
- An e-HealthCare scenario which contains 12 services.

The maximum number of parameters used to describe functional input and output parameters of services is 3. The number of instances (concepts) used to define the input (output) parameters of the service goal is 4. The execution times (ms) of the discovery process and the hole process for the composition of the three scenarios are respectively: 0.63; 1.32; 0.39.

First of all, this is a decentralised approach which composes a small number of web services (50 services maximum as the author said). This limitation is due to the growing size of the single Semantic Link Matrix SLM which must be computed for each client request.

SLM is used to organise the web services according to a logical dependency among input and output parameters of different web services. To deal with a large number of web services, the author assumes that a process of discovery has been performed in order to first retrieve a finite set of web services. Finally this model cannot deal with dynamic objects.

In contrast, our decentralised multi-agent approach can deal easily with a large number of web services, since agents link their services offline regardless of the client request. When the central agent receives the request, it broadcasts it to all available agents, that compute their local plans in parallel, which reduce tremendously the time of the composition process. Also, our multi-agent model can deal with dynamic domains and services can create and eliminate objects from the planning domain.

By comparing the empirical results we can see the effectiveness of our approach. We will call $r4$ the results for the problem P_4 from our approach and rT the results of the Telecom example from [25].

The two problems take nearly the same time to find the solution ($0,68$ s in $r4$ and $0,63$ in rT) when the number of services is nearly the same (30 in $r4$ and 35 in rT), but one can notice that our request (14 objects with 20 predicates in the initial and the goal states) is much more complex than the request considered in rT (3 instances in the initial state and one concept in the goal). Also, since our model deals with dynamic objects and extended services, the execution of a service

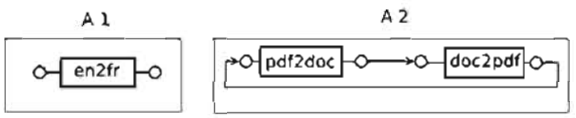

Fig. 9. Agent of web services.

by the planner and the comparison between two states is more complex than in rT.

5.6. Discussion

Although the multi-agent approach based on local heuristic has proved its efficiency to scale up the state-of-the-art of the web services composition techniques, it is not able to find solutions for all composition problems. We give here a simple example to demonstrate.

Let us take the two web services agents illustrated in Fig. 9.

The decentralised algorithm given is complete if the agents are independent. To illustrate this idea let us see the execution of the algorithm in a domain of web services constituted from two agents: $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1$ contains one service $en2fr$ to translate a *doc* file language from *english* to *french*. $en2fr$ is defined as follows:

- $Pin = \{\}$.
- $Pout = \{\}$.
- $Pinout = \{(F1 : file)\}$.
- $Prec = \{(doc\ F1), (en\ F1)\}$.
- $Effect = \{(doc\ F1), (fr\ F1)\}$.

$\mathcal{A}_2$ contains two services: $pdf2doc$ and $doc2pdf$.

Let the request $R = (init, goal)$ be defined as follow:

- $init = (\{(F1 : file)\}, \{(pdf\ \ F1), (en\ \ F1)\}$;
- $goal = (\{(F1 : file)\}, \{(pdf\ \ F1).(fr\ \ F1)\}$.

The distance between $init$ and $goal$ is:

$$distance(init, goal) = 2.$$

These are the responses of each agent when receiving R by applying the forward strategy:

- $\mathcal{A}_1$ does not have any executable plan so $\pi_1^1 = \emptyset$;
- $\mathcal{A}_2$ has only one executable plan which is $\langle\{pdf2doc[F1]\}\rangle$. When executing it, the newly reached state is $init_1 = (\{(F1 : file)\}, \{(doc\ F1), (en\ \ F1)\}$ The distance between $init_1$ and $goal$ is 4. So $distance(init_1.goal) = 4 > distance(init, goal) = 2$. In this case, agent $\mathcal{A}_2$ ignores this plan which takes away from the $goal$ and $\pi_2^1 = \emptyset$.

After this initial planning phase, no agent has proposed a plan for achieving the $goal$ and the system results in a failed state. Nevertheless, there exists a plan which is defined by:

$$\{pdf2doc[F1], en2fr[F1], doc2pdf[F1])\}.$$

Hence the algorithm based on local heuristic is not complete. The examples show that the method discards states that are further away from the goal than the current state. It may be the case, however, that we need to move away from the goal in order to reach it as illustrated above. To overcome this incompleteness, we extend the approach with a global heuristic to ensure the completeness of the algorithm.

6. Multi-agent approach using global heurisitic

In this section, we extend the previous multi-agent approach with a global heurisitic to ensure its completeness. The global heuristic is based not only on a local evaluation of the best promised plan to reach the goal, but also on a globally distributed estimation which guides the planner of an agent to choose the best local plan, guaranteeing the completeness and the optimality[3] of the algorithm. An agent computes the global heuristic associated with each service s once for each request, in a *backward* manner from the $goal$ to the $init$ state. We will first give some intuition behind the algorithm that will be sketched in the next sections.

In the next sections, we first give the basic idea of the global heurisitic (Section 6.1) and the algorithm to compute it (Section 6.2). Then we explain how to extract the best local plan from an agent graph based on the global heuristic (Section 6.3). and the multi-agent coordination strategy (Section 6.4). Finally, we study the complexity and the completeness of the approach based on global heuristic (Section 6.7) and discuss its empirical results (Section 6.8).

6.1. Basic idea

The global heuristic still consists of estimating the distance from a local state q_i^k of the agent $\mathcal{A}_i$ to the goal state $goal$ but now considering the best local plans of other agents. Hence the distance is computed in a distributed way.

[3]The optimality criterion is: minimising the number of services in the plan.

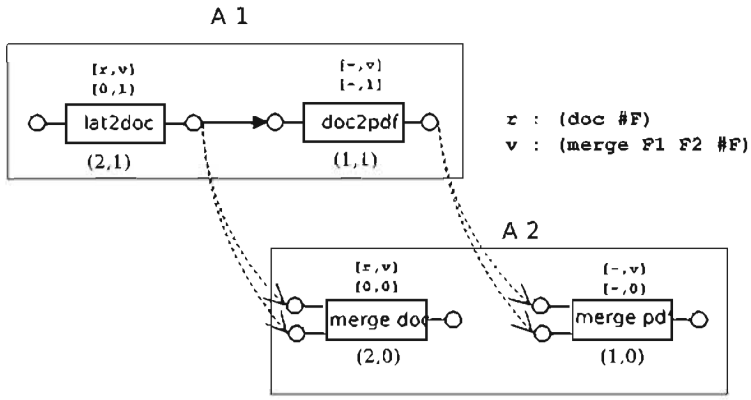

Fig. 10. Global heuristic computing.

First, as before, the agent $\mathcal{A}_i$ computes locally the distance from q_i^k to the closest node to the goal, $distance(q_i^k, q_i^{k'})$, where $q_i^{k'} = \pi_i(q_i^k)$ (result of applying the partial plan π_i to the local state q_i^k) and π_i is the best local partial plan. Then, each other agent $\mathcal{A}_{\alpha_j}$ can compute its local distances $distance(q_{\alpha_j}^l, q_{\alpha_j}^{l'})$ between all pairs of states $q_{\alpha_j}^l$ and $q_{\alpha_j}^{l'}$ of all its local states.

A propagation of such local distances from a local state q_i^k to neighbor agents is performed (as depicted in Fig. 10), using dependency links between agents (i.e. a link appears between a service s of agent $\mathcal{A}_{\alpha_j}$ and a service s' of agent $\mathcal{A}_{\alpha_{j+1}}$ when the preconditions of an input parameter of s' are included in the effects of an output parameter of s) to get a global heuristic measure (called *Distance*). This global heuristic measure is then given by:

$$Distance(q_i^k, q_{goal}) = distance(q_i^k, q_i^{k'}) + \sum_{\alpha_j \in I/i} distance(q_{\alpha_j}^l, q_{\alpha_j}^{l'})$$

where: q_{goal} is the goal state, $q_{\alpha_j}^{l'} \equiv q_{\alpha_{j+1}}^l$, i.e. the final state of agent A_{α_j} has a direct external link to the initial state in agent $A_{\alpha_{j+1}}$ and I denotes the set of agents indices $\{1, \ldots, n\}$.

The best local plan π of an agent A_i is thus the one leading to a state $q_i^{k'}$ such that $\pi(q_i^k) = q_i^{k'}$ and $\pi = \arg\min_{q_i^k} \pi_i(q_i^k)$.

6.2. *Global heuristic algorithm*

The global distributed heuristic of a service s is based a on distance measure which consists of two criteria which are:

- P: the promised degree to achieve the goal when using s;
- O: the promised degree to obtain the optimal solution when using s.

Initialisation A *colour* c_i (designated by a number) is assigned to each goal predicate $gp_i \in goal$ in the request $R = (init, goal)$. Two vectors $V_c[s]$ and $V_d[s]$ are assigned to each agent service s. Both vectors have a dimension equal to the number of predicate colours c_i. The type of vector $V_c[s]$ is boolean and indicates if s is coloured with c_i[4]. $V_d[s]$ contains the number of intermediate services needed to colour s with c_i.

We first compute for each predicate goal gp_i all the *producer* services CS^i of this predicate (i.e. those which have the predicate gp_i as an effect of their execution). Then, each service s_i^j in CS_i is coloured with the colour c_i. Each coloured service computes its links with services of other agents in a manner similar to that defined in Definition 4. This is repeated recursively until no links can be created.

Dissemination of colours After the initialisation, each colour c_i is disseminated as follows: for each service s predecessor of a coloured service s_c, s is coloured with c_i if all edges leaving s enter in a service coloured with c_i; then $V_d[s]$ is assigned the value $V_d[s_c] + 1$.

Computing promised degrees Finally, for each service s, the promised degree to achieve the goal by using s is $P(s) = \sum_i V_c[s](i)$, i.e. the number of predicate goals that are expected to be satisfied. The

[4] For the sake of readability, we will instead show V_c in the next example with the index i explicitly written in *true* cells, and '_' written in *false* cells.

promised degree to reach the optimal solution by using s is $O(s) = \sum_i V_d[s](i)$, i.e. the expected number of services to execute before reaching the goal. As it might be easily guessed, the algorithm will aim at maximising the P and minimising V.

After computing promised degrees, each agent erases the colour of its services and keeps only the degrees $P(s)$ and $O(s)$ of each service s.

6.3. Best local plan extraction

The extraction of the best local plan using global heuristics is as described in Section 5.1 with two extensions:

- Extending the condition of developing neighbor services of a service s (Section 5.1.1).
  The condition is that all incoming edges leave a colored service and $P(s) \geq max(P(Pred(s)))$ where $Pred(s)$ are all predecessor services of s.
- Extending the criteria used to extract the best partial plan (Section 5.1.2).
  When sorting the states based on their heuristics, if there are two states q_1, q_2 having the same $distance$ to $goal$, then the state having a path that maximises the promised degree P to reach the $goal$ is preferred. The shortest path of services between the initial state and the closest state to the goal state is computed based on the promised degree of services $O(s)$ rather than the current actual number of services in the plan. Finally, the state having the shortest path to $init$ is preferred.

6.4. Multi-agent coordination

In this section, we explain the coordination strategy between agents by extending the failure condition of the distributed algorithm and the merged strategy of the agents' plans based on the global heuristic.

In the complete approach, a *necessary condition* that a solution plan exists is that the union of colours of executable services in $init$ is equal to the set of all the $goal$ predicates colours C. Let S_{init} be the set of all executable services on $init$ coloured by any $goal$ predicate colour c_i.

Formally, $\bigcup_{s \in S_{init}} colours(s) = C$.

When this condition is verified, the selection of the best partial plan in a planning phase is made as follows: each agent $\mathcal{A}$ is supposed to record all intermediate states $init_i$ for each planning session (defined by a request $R = (init, goal)$).

Let $\langle q_1, \ldots, q_k, \ldots, q_l \rangle$ be the sorted set of all reachable states in the instantiation graph (Section 5.1.1) when receiving $init_i$, and let $p_1, \ldots, p_m$ be the associated paths to reach the states respectively.

During the planning phases, an agent can receive the same intermediate states $init_i$ for a multiple times. To avoid deadlock, the agent will not always propose its best partial plan $p1$, but a worse plan which helps reach the goal. To choose from the set of less plans, we use an indicator k which is equal to the number of times that $\mathcal{A}$ receives $init_i$ in the planning session. So when $\mathcal{A}$ receive a state $init_i$ for the first time, it will propose its best local plan $p1$, and so on it proposes p_k when receiving $init_i$ for the k-th times. If $k > m$, then $\mathcal{A}$ does not have a plan to propose. This idea will be explained in detail in Example 6.6.

When $\mathcal{A}_c$ receives agent plans $Plans = \{\pi_1, \ldots, \pi_n\}$, it divides it into two sets:

- $Plans_r = \bigcup_{\pi \in Plans} \{\pi / heuristic(\pi) \leq distance(init_i, goal)\}$, those are the plans which allow to approach the $goal$;
- $Plans_e = \bigcup_{\pi \in Plans} \{\pi / heuristic(\pi) \geq distance(init_i, goal)\}$, those are the plans that drive away from the goal.

If the set $Plans_r \neq \emptyset$, then $\mathcal{A}_c$ sorts $Plans_r$ and merge them to obtain a global partial plan Π_r. Π_r is then applied to $init_i$ to obtain a new state $init_{i+1}$. If $init_{i+1} = goal$ then a solution is found. If $Plans_r = \emptyset$ and the necessary condition that a solution exists is satisfied, this mean that we must temporarily go away from the $goal$ to allow for another agent to reconcile or achieve it later.

For that purpose, $\mathcal{A}_c$ sorts $Plans_e$ in ascending order to obtain $\langle \pi_1, \ldots, \pi_j, \ldots, \pi_l \rangle$ and apply the first plan π_1. Finally, the failure case is expressed by $Plan_r = Plan_e = \emptyset$. This strategy of partial plans selection avoids the sinks in the graph.

6.5. Example

In this section, we give an example to illustrate the execution of the planning algorithm based on global heuristics. Suppose that we have two agents: $\mathcal{A}_1$ contains services $lat2doc$ and $doc2pdf$, $\mathcal{A}_2$ contains two services: $mergedoc$ and $mergePdf$. Let $R = (init, goal)$ where:

- $init = \{(F1 : file), ((F2 : file))\}$, $\{(lat\ F1), (doc\ F2)\}$;
- $goal = \{(\#F : file)\}, \{(doc\ \#F),$ $(merge\ F1\ F2\ \#F)\}$.

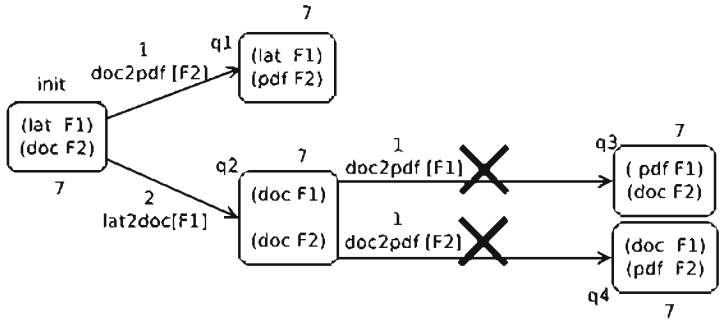

Fig. 11. Instantiation graph of $\mathcal{A}_1$.

The graph of services is given in Fig. 10. $\mathcal{A}_c$ distributed R to $\mathcal{A}_1$ and $\mathcal{A}_2$.

Initially, the list of colours $[r, v]$ (red and violet) is associated with *goal* predicates which are $(doc\ \#F)$, $(merge\ F1\ F2\ \#F)$. Each agent colours its services which produce each *goal* predicate with associated colour, so: $\mathcal{A}_1$ colours $lat2doc$ by (r). $\mathcal{A}_2$ colours $mergePdf$ by (v) and $mergeDoc$ by (r, v).

Second, the value (0) is associated with each colour predicate pg_i in V_d which contains the number of intermediary services required for colouring s with c_i. Then, colours are broadcasted for the pairs of vectors V_c and V_d of each service.

For example, (v) is propagated back from $mergePdf$ to $doc2pdf$ to be coloured by (v) as a result of external links, so the degree V_d of service associated with the colour (v) is equal to (1).

Colours (r, v) are propagated from $mergedoc$ back to $lat2doc$ to be coloured by (r, v).

Regarding the degree $V_d[r]$ of $lat2doc$; since $V_d[r]$ was equal to (0) and with the spread of colour (r) via $mergedoc$, the degree $V_d[r]$ becomes (1), the agent takes the minimum value to colour his service $lat2doc$ by (r). Hence $V_d[r] = 0$.

The necessary condition for the existence of a plan solution is verified for this problem. In fact, executable services in the $init$ state are $\{lat2doc, doc2pdf\}$. Its services are coloured by the *goal* predicates colours (r, v). After the verification of this condition, each agent deletes the colours of its services and keeps only the degrees $P(s)$ and $O(s)$ of each service s. Finally, the pair (P, O) (written in the figure close to each service box) is computed based on V_c and V_d.

Now, we illustrate the planning phases:

First iteration

The responses of the agents to R are:

- $\mathcal{A}_1$: in $init$, $lat2doc$ is executable on $F1$ and $doc2pdf$ is executable on $F2$;

  Hence after the instantiation of the executable graph, we obtain the graph in Fig. 11. During the instantiation of the graph, it should be noted that the service $doc2pdf$ successor service $lat2doc$ is omitted because $P(doc2pdf) = 1 < P(lat2doc) = 2$.

  By computing the distances between each state of the instantiation graph and the *goal* state, we find that there are two states with the same distance that is equal to 7. The plan for each state is:

  * $\langle doc2pdf[F1]\rangle$: which reach $q1$. The degree to reach the *goal* through this path is $P(doc2pdf|F1|) = 1$;
  * $\langle lat2doc[F2]\rangle$. The degree to reach the *goal* through this path is $P(lat2doc[F2]) = 2$;

  In this case, the agent chooses the path that maximises P, so $\pi_1 = \langle lat2doc[F2]\rangle$;
- $\mathcal{A}_2$: no service is executable, so $\pi_2 = \oslash$.

Because there is one partial plan executable on $init$, the global partial plan is $\Pi_1 = \langle lat2doc[F2]\rangle$.

Then $\mathcal{A}_c$ computes $\Pi_1(init)$ to obtain $init_1 = (\{(F1 : file), ((F2 : file))\}, \{(doc\ F1), (doc\ F2)\}$;

From $init^1$ and $goal$, $\mathcal{A}_c$ constructs a new request $R_1 = (init_1, goal)$, which will be distributed to all agents and save the partial plan from $init$ to $init_1$.

Second iteration

The response of agents to R_1 is:

- $\mathcal{A}_1$: the only executable service is $doc2pdf$, where the best local plan is $\pi_1 = \{doc2pdf|F1|, doc2pdf|F2|\}$;
- $\mathcal{A}_2$, the only executable service is $mergedoc$, the best local plan is $\pi_2 = \{mergedoc[F1, F2]\}$ which reaches the *goal* state.

When receiving both plans, the central agent notices that the plan π_2 has 0 as heuristic value, so $\Pi_2 = \pi_2$.

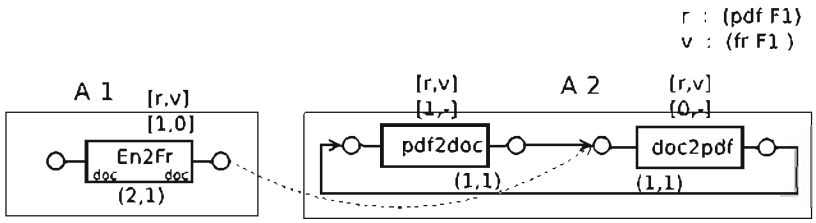

Fig. 12. Web services agent.

In this phase, our system found the solution plan which is extracted by the central agent: the overall plan is $\Pi = \langle lat2doc[F2], mergedoc[F1, F2]\rangle$.

6.6. Example II

Now we recall the example introduced in Section 5.6 and show how the decentralised algorithm using global heuristic can find its solution.

Given the query $R = (init, goal)$ defined by:

- $init = [\{(F1 : file)\}, \{(pdf\ \ F1), (en\ \ F1)\}]$;
- $goal = [\{(F1 : file)\}, \{(pdf\ \ F1), (fr\ \ F1)\}]$.

The computation of the global heuristic corresponding to the request is illustrated in Fig. 12.

The necessary condition to find a solution plan is verified for this problem. In fact, the service $\{en2fr\}$ is executable in $init$ and is colored by r and v, which are the colors of the $goal$ predicates.

Since the three services have the same degree P to achieve the $goal$, the extraction of the best partial plan is computed by using the local heuristic. In contrast, the merger strategy conducted by the central agent will be different in this example.

These are the responses of each agent when receiving the request R by applying services from $init$ to $goal$:

First iteration

- $\mathcal{A}_1$ does not have any executable service in $init$. So $\pi_1 = \emptyset$;
- $\mathcal{A}_2$ has two executable partial plans:
  - $\pi_2^1 = \{pdf2doc[F1]\}$.
    By applying this plan, the new reached is $init_1^1 = [\{(F1 : file)\}, \{(doc\ F1), (en\ F1)\}]$. The distance between $init_1^1$ and $goal$ is 4. So $distance(init_1^1, goal) = 4 > distance(init, goal) = 2$.
  - $\pi_2^2 = \emptyset$. In this case, the agent does not change the current state $init$. The distance between $init$ and $goal$ is 2.

  In this case, the agent $\mathcal{A}_2$ sorts the plans in ascending order. It obtains $\langle \pi_2^2, \pi_2^1\rangle$. Since it is the first time that $\mathcal{A}_2$ receives the intermediate state $init$, it sends to the central agent the plan π_2^2.

Since the central agent has receive two empty partial plans, the result of their merge is an empty plan and the new request $R1 = (init, goal)$ is broadcasted to the agents since the necessary condition to find a solution plan is verified.

The responses of the agents after receiving the request $R1$ are:

Second iteration

- $\mathcal{A}_1$ have one partial plan $\pi_1 = \emptyset$ executable in $init$. Since it has already proposed when receiving the $init$ in the previous phases, it will respond to the central agent that it does not have any plan to propose;
- $\mathcal{A}_2$ has two partial plans:
  - $\pi_2^1 = \{pdf2doc[F1]\}$, with $heuristique(\pi_2^1) = 4$;
  - $\pi_2^2 = \emptyset$.

  since $\mathcal{A}_2$ has received the $init$ state for the second time, it will respond by the plan having the second position in the sorted list of plans $\langle \pi_2^2, \pi_2^1\rangle$. Hence, it sends to the central agent the second plan π_2^1.

Since the central agent does not receive any partial plan which allow to approach the $goal$, then $Plan_r = \emptyset$. In this case, the central agent chooses the best plan from $Plans_e$ which drive away from the $goal$. So $\Pi_1 = \pi_2^1 = \{pdf2doc[F1]\}$.

Afterward $\mathcal{A}_c$ computes $\Pi_1(init)$ and obtain $init^1 = |\{(F1 : file)\}, \{(doc\ \ F1), (en\ \ F1)\}|$; from $init_1$ and $goal$, $\mathcal{A}_c$ build a new request $R^1 = (init_1, goal)$, which will be broadcasted to all agents and save the partial plan form $init$ to $init_1$.

Third iteration

The responses of the agents when receiving R_1 are:

- $\mathcal{A}_1$ has two plans to propose:
  - $\pi_1^1 = \emptyset$ with (4) as heuristic;
  - $\pi_1^2 = \langle\{en2fr[F1]\}\rangle$ with (2) as heuristic.

  So π_1^2 which has the minimal heuristic is sent to the central agent.
- $\mathcal{A}_2$ has two plans to propose:
  - $\pi_2^1 = \emptyset$ with (4) as heuristic;
  - $\pi_2^2 = \langle\{doc2pdf[F1]\}\rangle$ with (2) as heuristic.

  So π_2^2 which has the minimal heuristic is sent to the central agent.

The two plans received by the central agent allow to approach the *goal*. Thoses plans are not mergeable. In fact:

- $prec(\pi_1^2) = \{(enF1), (docF1)\}$;
- $prec(\pi_2^2) = \{(docF1)\}$.

So, $prec(\pi_1^2) \cap prec(\pi_2^2) = \{(docF1)\}$.

Since the two plans have the same heuristic (2), the central agent chooses the plan maximising P to reach the *goal* which is π_1^2 ($P(\pi_1^2) = 2$ and $P(\pi_2^2) = 1$).

So, $\Pi_2 = \pi_1^2 = \langle en2fr[F1]\rangle$.

The new intermediate state is

$$init_2 = [\{(F1 : file)\}, \{(doc\ F1), (fr\ F1)\}].$$

Then $\mathcal{A}_c$ builds a new query $R_2 = (init_2, goal)$, to be distributed to the agents and saves the partial plan from $init_1$ to $init_2$.

Fourth iteration

The agents responses to R_2 are:

- $\mathcal{A}_1$ has only $\emptyset$ to propose;
- $\mathcal{A}_2$ has one partial plan that reach the *goal*. This plan is $\pi_2 = \langle\{doc2pdf[F1]\}\rangle$. This plan is directly send to the central agent.

The agent $\mathcal{A}_2$ informs $\mathcal{A}_c$ that it finds the *goal*, so $\Pi^3 = \langle\{doc2pdf[F1]\}\rangle$. $\mathcal{A}_c$ extracts the solution plan $\langle\Pi^1, \Pi^2, \Pi^3\rangle$ = $\langle pdf2doc[F1], en2fr[F1], doc2pdf[F1]\rangle\rangle$.

6.7. *Complexity and completeness*

6.7.1. *Complexity*

The complexity of the complete algorithm is equal to the sum of the complexity of computing the global heuristic (Section 6.2) and the complexity of the algorithm based on local heuristics (Section 5.4). From Section 5.4, the latter is polynomial both in the number of agents n and in the number of services per agent i_S.

Let m_c be the number of colours in the *goal* predicates, and m the maximum number of services per agent.

The complexity of computing the global heuristics is equal to the sum of the complexity of:

- Computing the service producer of the *goal* predicates which is equal to $\mathcal{O}(m_c * n * m)$;
- The dissemination of colours which is equal to $\mathcal{O}(m_c * n^2 * m^2)$;
- Computing the promised degrees which is equal to $\mathcal{O}(2 * n * m_c)$;

This complexity is polynomial, so the complexity of the complete algorithm is also polynomial.

6.7.2. *Completeness*

The decentralised algorithm based on global heuristic is complete and optimal. In fact, in the worst case, each agent proposes a plan to build a single service which ensures that every state will be visited including the goal state.

Proof. Let $C = (WS_1, \ldots, WS_n)$ the community of web services, $A_1, \ldots, A_n$ the set of associated agents, $R = (init, goal)$ the request. Our proof by induction is:

if $\Pi = \langle\pi_1\rangle$

since the problem is resolvable by a single partial plan, thus *init* and *goal* states are partitionable as follows: $init = \{init_1, \ldots, init_n\}$ and $goal = \{goal_1, \ldots, goal_n\}$, where:

- $init = \{init_1 \cup \cdots \cup init_n\}$ and $init_1 \cap \cdots \cap init_n = \oslash$, $init_i \subset prec(services(A_i))$.
- $goal = \{goal_1 \cup \cdots \cup goal_n\}$ and $goal_1 \cap \cdots \cap goal_n = \oslash$ and $goal_i \subset effect(services(A_i))$.

In this case, when the central agent broadcasts R to the set of agents, each agent A_i will propose a partial plan p_i^1 that is the optimal local plan from $init_i$ to $goal_i$. The local algorithm is complete since it selects its local optimal plan from all the possible local plans constructed by using all un-pruned local services (pruned services are omitted because the local plan cannot reach its goal by using them). Also, the local plan is optimal because it uses the promised degree for the services ($O(s)$) to compute the shortest path between *init* and *goal* which is the optimal solution.

Since the preconditions of the partial plans are independent, the partial plans are completely mergeable and $\pi_1 = merge(\pi_1^1, \ldots, \pi_n^1)$ is the solution.

if $|\Pi| > 1$

Now suppose that our algorithm is able to find all the partial plans $\langle \pi_1, \ldots, \pi_{m-1} \rangle$ constructed by $m-1$ iteration of the algorithm. We should prove that it is also able to find the last partial plan π_m.

In fact, after executing $m-1$ iterations of the algorithm, the plan obtained will be $\Pi_{m-1} = \langle \pi_1, \ldots, \pi_{m-1} \rangle$. When Π_{m-1} is applied on $init$, the result will be $init_{m-1}$. When the central agent broadcasts the new query $R_{m-1} = (init_{m-1}, goal)$ each agent A_i computes its optimal local plan p_i^m based on the promised degrees (degree to achieve the goal and to reach the optimal solution) computed initially by using the global heuristic algorithm explained in Section 6.2.

For the same reasons explained when the size of the global plan is 1, the set of plans $\langle \pi_1^m, \ldots, \pi_n^m \rangle$ proposed by the agents are mergeable, so $\pi_m = merge(\pi_1^m, \ldots, \pi_n^m)$. Also when the last partial plan π_m is added to $\langle \pi_1, \ldots, \pi_{m-1} \rangle$, the global plan obtained $\langle \pi_1, \ldots, \pi_{m-1}, \pi_m \rangle$ is optimal since the computation of π_m takes into account the optimality by using the promised degree of services to reach the optimal solution.

The algorithm is complete and optimal by induction. □

6.8. *Extending the implementation with the global heuristic*

The architecture developed in Section 5.5 is extended to implement the complete distributed algorithm. So, each agent joining the community of web services will create not only the links between its services, but also links with the other agents' services. When a client submits a request to the system, and before starting the planning phases, each agent must compute the promised degrees (see Section 6.2) of all its services.

Table 2 shows the comparison between the execution time of decentralised incomplete (I-MA) and decentralised complete (C-MA) algorithms. For each request, we give also the number of objects ($|Obj|$) and the number of predicates ($|pred|$) in the query for the initial (In) and the goal (Go) states, and the number of services in the plan solution ($|\Pi|$).

Empirical results show that the time execution is almost the same for both decentralised algorithms. Complete algorithm takes a few milliseconds more. This is the time to compute the global heuristics.

Table 2
Empirical results

Query	|Obj|		|Pred|		Exec time (s)		Plan
	In	Go	In	Go	I-MA	C-MA	$\|\Pi\|$
P3	7	3	12	10	0.69	0.74	13
P4	14	14	20	20	0.68	0.75	18
P5	9	5	15	8	–	0.91	13
P6	14	7	23	14	–	1.1	23

In contrast, the experiments of the algorithm based on global heuristic shows its ability to find solutions when the the algorithm based on local heuristic is not ($P5$ and $P6$). Experiments show also that the distributed complete algorithm can find the solution for complex problems in a short time. For example, in (1.1) second, the algorithm finds the solution for the problem $P6$ which contains 14 objects in its initial state and require the composition of 23 services to find the optimal solution.

7. Usefulness and applicability of the approach in multiple domains

In order to show the usefulness and applicability of our distributed approach in domains other than text processing, we present another example which deals which deals with travel web services.

Let us consider three web services which are intended to reserve train or plane tickets.

1. Web service WS_1 is a train service. It has two services: $(Toulouse \rightarrow Lyon)$ and $(Lyon \rightarrow Paris)$ to reserve tickets to transport persons from $Toulouse$ to $Lyon$, and $Lyon$ to $Paris$. Other services could be considered in this web services such that $(Lyon \rightarrow Rouen)$, etc.
2. Web service WS_2 is a plane service. It has one service: $(Paris \rightarrow Washington)$ to reserve tickets to transport persons from $Paris$ to $Washington$. Other services could be considered in this web services such that $(Paris \rightarrow Miami)$, etc.
3. Web service WS_3 is a plane service. It has three services: $(Washington \rightarrow Chicago)$, $(Chicago \rightarrow San - Francisco)$ and $(Chicago \rightarrow Los - Angeles)$ to reserve tickets to transport persons from $Washington$ to $Chicago$, from $Chicago$ to $SanFrancisco$ and

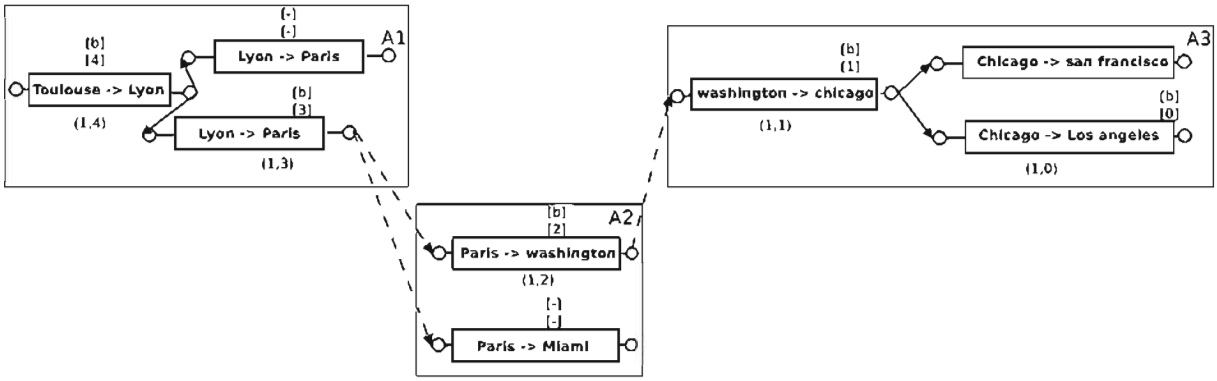

Fig. 13. Graph of travel web services.

from $Chicago$ to $LosAngeles$. Other services could be considered in this web services.

As an example, let us suppose that we have a person who wants to reserve a ticket to travel from $Toulouse$ to $LosAngeles$. In this case the request is defined by $R = (init, goal)$ where:

- $init = (\{(Ps : Person), \{(Toulouse\ Ps)\}\})$,
- $goal = (\{(Ps : Person), \{(Los{-}Angeles\ Ps)\}\})$.

The travel service $(Toulouse \rightarrow Lyon)$ can be defined as follows:

- $Pin = \{\}$.
- $Pout = \{\}$.
- $Pinout = \{(Ps : Person)\}$.
- $Prec = \{(Toulouse\ Ps)\}$.
- $Effect = \{(Lyon\ Ps)\}$.

All the other services can be defined in the same way.

The graph of services is given in Fig. 13. $\mathcal{A}_c$ distributes R to $\mathcal{A}_1$, to $\mathcal{A}_2$ and to $\mathcal{A}_3$.

Initially, the colour $[b]$ (blue) is associated with the only predicate in the $goal$ state which is $(los - angeles\ Ps)$. Each agent colours its services, which produce the $goal$ predicate, with blue. $\mathcal{A}_3$ is the only agent having service $(chicago \rightarrow losangeles)$ which produces $(losAngeles\ Ps)$, so $(chicago \rightarrow losAngeles)$ is coloured by (b) ($V_c = [b]$ and $V_d[b] = [0]$).

Second, the colour (b) is propagated back from service: $(chicago \rightarrow losAngeles)$ to service: $(washington \rightarrow chicago)$ to be coloured by (b) ($V_d[b] = [1]$), and from service: $(washington \rightarrow chicago)$ to service: $(Paris \rightarrow washington)$ as a result of external links ($V_d[b] = [2]$) and so on.

The necessary condition for the existence of a plan solution is verified for this problem. In fact, the executable service in the $init$ state $(Toulouse \rightarrow Lyon)$ is coloured by the $goal$ predicate colour (b).

After the verification of this condition, the pair (P, O) (written in the figure close to each service box) is computed based on V_c and V_d.

Now, we illustrate the planning phases:

First iteration

The responses of the agents to R are:

- $\mathcal{A}_1$: in $init$, two plans are executable:
  1. $\pi_1^1 = \langle Toulouse \rightarrow Lyon[Ps] \rangle$.
  2. $\pi_1^2 = \langle Toulouse \rightarrow Lyon[Ps], Lyon[Ps] \rightarrow Paris[Ps] \rangle$.

  $\pi_1^2[init] = init_1^1 = (\{(Ps : person), \{(Lyon\ Ps)\}\})$ and $\pi_1^2[init] = init_1^2 = (\{(Ps : person), \{(Paris\ Ps)\}\})$. Since $distance(init_1^1, goal) = distance(init_1^2, goal) = 2$ and $P(\pi_1^1) = P(\pi_1^2) = 1$; in this case, the agent chooses the path that minimises O. Since $O(\pi_1^1) = 4$ and $O(\pi_1^2) = 3$, the best partial plan is $\pi_1 = \pi_1^2$.
- $\mathcal{A}_2$: no service is executable, so $\pi_2 = \oslash$.
- $\mathcal{A}_3$: no service is executable, so $\pi_3 = \oslash$.

Because there is one partial plan executable on $init$, the global partial plan is $\Pi_1 = \pi_1^2 = \langle Toulouse \rightarrow Lyon[Ps], Lyon[Ps] \rightarrow Paris[Ps] \rangle$.

Then $\mathcal{A}_c$ computes $\Pi_1(init)$ to obtain $init_1 = (\{(Ps : person), \{(Paris\ Ps)\}\})$;

From $init^1$ and $goal$, $\mathcal{A}_c$ constructs a new request $R_1 = (init_1, goal)$, which will be distributed to all agents and save the partial plan from $init$ to $init_1$.

In the second iteration and the third iteration, the global partial plan are respectively $\Pi_2 = \langle Paris \rightarrow washington[Ps] \rangle$ and $\Pi_3 = \langle washington \rightarrow$

Table 3
Empirical results

Query	People number	I-MA	C-MA	$\|\Pi\|$
P7	1	0.17	0.15	5
P8	3	–	0.33	13
P9	7	0.61	0.53	25

$chicago[Ps], chicago \rightarrow los-angeles[Ps]\rangle$. Finally, the obtained plan is formed by $\langle \Pi_1, \Pi_2, \Pi_3 \rangle$ leading to the *goal*.

7.1. Empricial results

In this section, we give the results of our experimentations in the travel domain. This domain is defined by a set of travel web services like Section 7. The domain contains seven web services. Six of them are dedicated to reserve trains between cities in a country (*France*, *Spain*, *Germany*, *Greece*, *Italy* and the *United Kingdom*) The seventh web service is dedicated to reserve planes between the airports of the six countries. The number of train services is in $[3, 10]$ per country. The number of services in the plane web services is 12. The results of three queries are given in Table 3. For each query, we give respectively the number of people who want to travel between multiple cities, the execution time for decentralised incomplete (I-MA) and decentralised complete (C-MA) algorithms. Those results show the effectiveness of the approach to find a solutions in about one second in multiple complex problems.

More sophisticated scenarios can of course be designed in such a domain, using the full expressiveness of our model, through preconditions and effects that will represent the links between services provided by distinct web services. In fact, almost all kinds of domains that are present today on the web can be expressed in such a way. For instance, our approach can be applied in a scenario of a user who wants to buy a book on internet (*amazon.com* for example); in this scenario, this request will be sent in a context where a postal web service and a bank web service are composed to respond to the request. Our distributed approach can also be applied in a scenario where a user wants to rent a movie, in MPEG format; in this scenario, this request will be sent in a context where no agent providing MPEG videos is available. However, there is instead a UIF video provider and a UIF-MPEG converter. Such scenario could be addressed by our approach.

8. Conclusion and future works

In this paper, we proposed a complete algorithm for the composition of web services which models the problem as a multi-agent planning problem. In this new approach, a planner agent is associated with each web services. Each planning agent builds from its services a graph to relate its homogeneous services. It also creates links between its own services and the services of all other agents in the planning domain. By doing so, the response time of the system is decreased, since a large part of planning computation is made offline and only once for all the requests. Agents collaborate by merging their best plans to reach the goal quickly. The central agent controls the collaboration among web services agents, and the dissemination of the requests to the agents. Generaly speaking, our approach is able to address problem of multi-agent planning where the planning problem of each agent is represented as a local graph.

From a modeling point of view, this work gives us a solid first step to introduce fault occurrences in web services, and integrate planning with diagnosis of such faults. Each agent has to monitor the execution of its plan and infer the current (possibly faulty) state of the plan execution, which will be used to repair the faults by re-planning.

Finally, we want to enhance our model through the notion of resource reservation: once a plan is found, one needs before executing it to reserve the actual services that are needed over the Internet, in order to avoid conflicts with other plans being concurrently executed over the community of web services.

References

[1] M. Pistore, P. Bertoli, F. Barbon, D. Shaparau and P. Traverso, Planning and monitoring web service composition, in: *Proc. of the International Conference on Automated Planning and Scheduling*, Vol. 3192 (2004), pp. 106–115.

[2] M. Pistore, P. Traverso and P. Bertoli, Automated composition of web services by planning in asynchronous domains, in: *Proc. of the 15th International World Wide Web Conference*, Vol. 174 (3–4), 2005, pp. 2–11.

[3] MB. Juric, B. Mathew and P. Sarang, Business Process Execution Language for Web Services: BPEL and BPEL4WS, Packt Publishing, Technical report, 270, 2004.

[4] M. Pistore and P Traverso, Planning as model checking for extended goals in non-deterministic domains, in: *Proc. of the 17th International Joint Conference on Artificial Intelligence*, Vol. 1, 2001, pp. 479–484.

[5] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri and P Traverso, MBP: A model based planner, in: *IJCAI'01 Workshop on Plan-*

ning under Uncertainty and Incomplete Information, Vol. 4, 2001, pp. 93–97

[6] A. Cimatti, M. Pistore, M. Roveri and P. Traverso, Weak, strong, and strong cyclic planning via symbolic model checking, *Journal of Artificial Intelligence* **147**(1–2), (2003), 35–84.

[7] A. Lazovik, M. Aiello and M. Papazoglou, Planning and monitoring the execution of web service requests, *International Journal on Digital Libraries* **6**(3), (2006), 235–246.

[8] A. Lazovik, Interacting with service compositions, PhD dissertation, University of Trento, 2006.

[9] B. Srivastava, Web service composition – current solutions and open problems, in: *ICAPS 2003 Workshop on Planning for Web Services*, Vol. 1, 2003, pp. 28–35.

[10] S. McIlraith and T. Son, Adapting Golog for composition of semantic Web services, in: *Proc. of the Eighth International Conference on Knowledge Representation and Reasoning*, Morgan Kaufmann Publishers, 2002, pp. 482–493.

[11] J. Levesque, R. Reiter, Y. Lesperance, F. Lin and R. Scherl, GOLOG: A logic programming language for dynamic domains, *Journal of Logic Programming* **31**(1–3), (1997), 59–83.

[12] R. Chinnici, M. Gudgin and J. Moreau, Web services description language, http://www.w3.org/TR/wsdl20/, Technical report, 2007.

[13] J. Peer, Web services composition as AI planning, Technical report, Universidad of Granada, 2005.

[14] J. Peer, Web services architecture overview: The next stage of evolution for e-business, Technical report, 2000.

[15] Y. Yan, Y. Liang and Y. Liang, Composing business processes with partial observable problem space in web services environment, in: *Proc of the Fourth IEEE International Conference on Web Services. NRC 48742*, 2006, pp. 541–548.

[16] M. Ghallab, D. Nau and P. Traverso, *Automated Planning. Theory and Practice*, Morgan Kaufmann Publishers, 2005.

[17] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld and D. Willkins, PDDL: The planning domain definition language, in: *IPS-98 Planning Competition Committee*, Technical report, 1998.

[18] D. Wu, E. Sirin, J. Hendler, D. Nau and B. Parsia, Automating DAML-S web services composition using SHOP2, in: *Proc. of the Second International Semantic Web Conference*, Vol. 2870, 2003, pp. 195–210.

[19] F. Lecue and A. Leger, A formal model for semantic web service composition, in: *Proc. of the Fifth International Semantic Web Conference*, Vol. 4273, 2006, pp. 385–398.

[20] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, Automated discovery, interaction and composition of semantic web services, *Journal of Web Semantics* 1(1) (2003), 27–46.

[21] M. EL Falou, M. Bouzid, A.-I. Mouaddib and T. Vidal, Automated web service composition using extended representation of planning domain, in: *Proc. of the Eighteenth European Conference on Artificial Intelligence*, Vol. 1, 2008, pp. 762–763.

[22] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta and K. Vahi, The role of planning in grid computing, in: *Proc. of the Thirteenth International Conference on Automated Planning and Scheduling*, Vol. 15, 2003, pp. 9–13.

[23] M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E Fink and J. Blythe, Integrating planning and learning: The PRODIGY architecture, *Journal of Experimental and Theoretical Artificial Intelligence* 7 (1995), 81–120.

[24] J. Hoffmann and B. Nebel, The FF planning system: Fast plan generation through heuristic search, *Journal of Artificial Intelligence Research* (2001), 253–302.

[25] F. Lecue, Web services composition: Semantic link based link approach, PhD dissertation, Ecole Nationale Superieur Des Mines de Saint-Etienne, France, 2008.

[26] ADETTI, D5.2: Service composition and execution in ip2p, Technical report of CASCOM project: Context Aware Business Application Service Co-ordination in Mobile Computing Environments, 2006.

[27] Y. Charif, Choregraphie dynamique de services basee sur la coordination d'Agents introspectifs, PhD dissertation, Universite Pierre et Marie Curie (Paris VI), 2007.

[28] D. Pellier and H. Fiorino, Un modele de composition automatique et distribuee de services web par planification, *Revue d'Intelligence Artificielle* 23(1) (2009), 13–46.

[29] M. EL Falou, M. Bouzid, A.-I. Mouaddib and T. Vidal, Automated web service composition: A decentralised multi-agent approach, in: *Proc. of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 1, 2009, pp. 387–397.

[30] R. Fikes and N.J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, in: *Proc. of the Second International Joint Conference on Artificial Intelligence*, Vol. 2(3–4), 1971, pp. 189–208.

[31] Z. Maamar, SZ. Quan and B. Boualem, Towards a conversation-driven composition of web services, in: *Proc. of the Web Intelligence and Agent Systems*, IOS Press, Amsterdam, The Netherlands, Vol. 2(2), 2004, pp. 145–150

[32] F. Lecue, A. Delteil and A. Leger, sslGolog: When conditional compositions of web services meet semantic links and causal laws, *Journal of Web Intelligence and Agent Systems*, IOS Press, 9(1) (2011), 1–25.