Monfroy, Eric; Perrin, Olivier; Ringeissen, Christophe

Modeling Web Services Composition with Constraints

Revista Avances en Sistemas e Informática, vol. 5, núm. 1, mayo, 2008, pp. 173-179

Universidad Nacional de Colombia

Medellín, Colombia

Modeling Web Services Composition with Constraints

Modelando la Composición de Servicios Web con Restricciones

Eric Monfroy, PhD[1], Olivier Perrin, PhD [2], Christophe Ringeissen, PhD [2]
[1]Universidad Técnica Federico Santa María, Valparaíso, Chile
LINA, Université de Nantes, Francia
[2]LORIA, Campus Scientifique, BP 239, 54506 Nancy Cedex, Francia
eric.monfroy@inf.utfsm.cl, {olivier.perrin, christophe.ringeissen} @loria.fr

Resumen—En este artículo abordamos el problema de composición de los servicios Web. Nuestra estructura se basa en la existencia de una composición abstracta, es decir, la forma como algunos servicios de diferentes tipos se pueden combinar entre sí con el fin de realizar una tarea determinada. Nuestro enfoque consiste en instanciar esta representación abstracta de un servicio Web compuesto, seleccionando los servicios web concretos más apropiados. Esta instanciación se basa en técnicas de programación de restricciones, las cuales nos permiten comparar los servicios Web de acuerdo a la consulta dada. La composición es realizada de manera incremental propagando algunas restricciones asociadas a los servicios Web. Entonces, la instanciación puede actualizarse dinámicamente durante la ejecución vía una fase de monitoreo. Nuestro enfoque está ilustrado en una composición Web, describiendo el comportamiento de un portal photolab. En este ejemplo, nosotros usamos técnicas basadas en restricciones para dinámicamente gestionar y calcular los tiempos de ejecución estimados (anunciados por los servicios) para cumplir a cabalidad o ejecutar a tiempo la tareas requeridas.

Palabras Clave—Servicios Web, Problema de la Composición, Razonamiento con Verificación de Restricciones.

Abstract—In this paper we consider the composition problem of Web services. Our framework is based on the existence of an abstract composition, i.e, the way some services of different types can be combined together in order to achieve a given task. Our approach consists in instantiating this abstract representation of a composite Web service by selecting the most appropriate concrete Web services. This instantiation is based on constraint programming techniques which allow us to match the Web services according to a given query. The composition is built in an incremental way propagating some constraints attached to Web services. Then, the instantiation can be dynamically updated during execution via a monitoring phase. Our approach is illustrated on a Web composition describing the behavior of a photolab portal. On this example, we use constraint techniques to dynamically manage and compute the estimated execution times (announced by services) to fulfill and achieve on time the requested tasks.

Keywords—Web Services, Composition Problem, Constraint Reasoning .

I. INTRODUCTION

Composition can be seen from various points of view. Many different techniques have been developed, including planning in AI [1], situation calculus [2,3,4], conversational transition systems [5], or symbolic model-checking applied to planning [6]. In this paper, we present an approach in which an abstract representation of a composition is already given and has to be instantiated by concrete Web services possibly interacting via complex protocols. In this context, we are looking for the orchestration implementing the abstract composition. We present a framework where constraint solving allows us to instantiate the composition with respect to input queries and properties of Web services. Our objective is to generate a concrete and executable service composition from a schematic (abstract) one.

Controling a composition can be very complex: one reason is the non-deterministic behavior of services; another reason is the possible failure of services involved in the composition. Therefore, exchanged messages are difficult to manage since they include complex data related to different aspects such as security, reliability, or presentation. The combination of all these aspects can generate a very complex design, and the resulting

code can be difficult to write, to maintain, and to adapt. Implementing a composition requires taking into account different aspects (such as control flow, data flow, security, reliability, ...) and constraints (such as compatibility, time for achieving tasks, costs, ...). Languages like WS-BPEL could allow us to implement a composition covering at the same time all those aspects. However, these aspects make the task very time consuming and error prone.

In this paper, we present an approach that is aspect-oriented, in the sense that it decouples the different aspects (for instance the temporal aspect, i.e., times required for achieving tasks or scheduling of these tasks with respect to requirements of the query or to some costs) of the component services, and in the sense that the composition is expressed separately from these aspects. We provide a constraint-based approach (see e.g., [7,8] for constraint programming and constraint reasoning references) that allows the automated generation of the composition with appropriate services. The resulting composition is compatible with the constraints defined in the respective aspect (temporal, reliability, security,...) and is dynamic in the sense that the composition can be automatically adapted (it can be re-instantiated) if one or more constraints are no more satisfied. It is easier to deal with the needed adaptations as the protocol is already defined, and the selection of services is independent from the interactions (messages exchange, message structure, ...). In our approach, the logic of the composition already exists, as the control flow is already defined. Therefore, the role of a solver is to find a set of services of the right types and right characteristics in order to generate an instantiation of the composition that fulfills all the requirements. Constraints are then considered to model additional properties of services.

The rest of the paper is organized as follows: in Section 2, we detail a possible scenario. Section 3 presents our constraint-based framework. In Section 4, we discuss related work, and Section 5 concludes.

II. MOTIVATING EXAMPLE

We illustrate our approach by a significant example detailed below. Let us consider a Web portal for printing digital pictures. The actors of the scenario are the clients, the portal, the labs in charge of printing the pictures, the delivery services, and the bank. A printed picture has several properties, but in our example, we only retain the following: its size, its quality, its price.

We consider given the composition that allows a client to send a query to a portal in order to print some pictures. This means that the workflow is already given, and that the process specifying the schedule of the different activities is known. However, we do not know which concrete services will make the work. To select these services, we use a constraint solver that will try to find the assignments (i.e., instantiation of the Web services). The starting point is the query of the client. In fact, the client is able to express some constraints (seen as preferences) to be satisfied by the composition. For instance, a client may want to get a set of printed pictures in at most 18 hours (delivery included). The type of queries that can be expressed by the client are:

- Print and send me these 100 pictures (no temporal or quality constraints),
- Print and send me these 100 pictures at minimum cost (no temporal constraints),
- Print and send me these 100 pictures at the average quality and minimum cost (no temporal constraints),
- Print and send me these 100 pictures at the average quality and minimum cost within 24 hours,
- Print and send me these 100 pictures at the 6x7 format and the 8,5x11 format as soon as possible.

Each actor also has a set of properties. For instance, the client can have some fidelity points that will give him (her) a discount. A photo lab can print different formats, but not necessarily all of them. It can grant some discounts with respect to the volume of work. The delivery service can have different levels of service (gold service for delivery in 24 hours, silver service in 36 hours, ...) and different qualities of printings.

Roughly, the different steps of the process are given as follows: the client will send a query to the portal service in order to get a quote. Then, if he accepts the proposition, he will send the files to be printed. The portal can treat the query, ask to photo lab services their prices given the preferences of the client, select the photo lab services that can match the query, send the result (estimates, propositions) to the client, and then send the files to the photo lab services. The photo lab services receives the query from the portal, check if there are delivery services compatible with this query, computes the time needed to print the pictures and to send them to the client, sends some requests to the bank service, prints the files, sends them to the delivery, ...The bank service receives the query from the photo lab service, checks the account, transfers the amount, and notifies the photo lab service. Then, the delivery service receives the query from the photo lab service, computes an estimate, receives the order, delivers the pictures.

In this process, the scheduling of tasks as well as types of each service (portal, photo lab, delivery, bank) are known. The objective is to instantiate this process with concrete services such that the constraints specified in the client query are satisfied. Suppose that, for each service, we associate temporal constraints: estimated execution times (min, max), availability periods, a combination of estimated execution times for given periods.

If the client queries a given task in a specified deadline, then the objective of the constraint solver is to find an enumeration of services whose types are compatible with those defined in the abstract composition, such that constraints are satisfiable. For instance, if the query is "I want my photos in the next 12 hours", and if we suppose we have 4 photo lab services, with 2 of them that take at least 36 hours to print the pictures, then

these two services will be out of the instantiation since they do not satisfy the temporal constraint of the query.
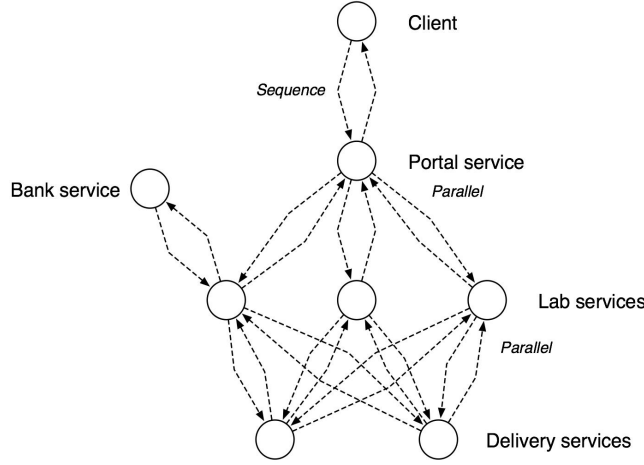

Figure 1: The process

This aspect is also interesting for the monitoring activity. For instance, let us suppose a constraint is no longer satisfied during the execution. We can adapt the flow by selecting another set of services satisfying the query. For instance, suppose a photo lab is out of its expected schedule. Given the horizontal constraints (constraints for building an instantiation as opposed to vertical constraints that model the abstract composition), we are able to monitor this state, and to select a delivery service that is able to deliver the photos in the expected time (e.g., using a "gold" service). In this case, we reschedule the flow defined during the instantiation step.

To summarize, our objectives are twofold. First, given all the constraints of services and the client query, we aim at finding one (or more) topology that satisfies the query, the requirements of the query, and the constraints induced by the chosen set of services.

Second, once a topology has been selected, and the corresponding services are currently executed, we verify that the execution is consistent with the previously defined constraints. If this monitoring phase reveals that a service does not satisfy a constraint anymore, a new instantiation of the abstract process is computed in order to maintain satisfiability of constraints. In case this is not possible, an approximate solution (i.e., violating some constraints) could be computed.

III. A CONSTRAINT-BASED APPROACH

In our constraint-based framework, we suppose that a composition is already defined in an abstract way. We know the different types of involved services, and all interactions between them. We consider this as a vertical composition, i.e., the abstract workflow specifies the different steps to achieve the objective.

This abstract representation provides us the topology of the composition (the protocol and the possible connections) and the types of services we need to instantiate. In our example, this means that we need to find a service (or several ones) having the type photo lab, and then a service having the type delivery in order to send the photos to the consignees. In this context, we are able to build an instantiation, i.e., a concrete representation of the composition, and to build and manage both the control and the data flow with respect to the constraints defined by the client and services from the community.

A   Constraints for the Composition Problem

We distinguish two kinds of constraints to express the composition problem. The first kind are constraints called vertical constraints. These constraints are needed to set up the abstract composition, i.e., the process (see Figure 1). For instance, constraints of this type are constraints such as "a delivery service can be called only if the caller is a photo lab service", "a portal can be connected to at most 20 photo lab services", "the photo lab service X can return me a proposal from at most Y services", or "the photo lab service must return a message to the portal service".

The second kind of constraints are called horizontal constraints. These constraints are needed to build an instantiation of the composite service, in terms of management of the data flow and the messages between the services. In this paper we focus on horizontal constraints.

We will use the horizontal constraints to select the concrete services and to instantiate the abstract composition. For instance, we will only select the photo lab services that are able to print in 8,5x11 format. This means that we may only select 4 services on the 10 photo lab services that exist in the community. If the query contains different formats to print, constraints may also be used to select different photo lab services, for instance one to print 4x6 format and the other one to print 8,5x11 format.

Horizontal constraints are used to manage the data flow. In fact, since the topology is already known, these constraints will be used by the solver to select the appropriate services of each type. Then, we can imagine different types of horizontal constraints in order to manage the different aspects of a composition, specifying temporal constraints as in our case or reliability/security constraints in a different context.

Horizontal constraints are used to instantiate the process but also to monitor it. Clearly, these constraints have an impact on the composition as illustrated by the following example: assume that we want 1000 photos in 4x6 format, and 200 photos in 8,5x11. Suppose that the lab which is able to print photos in the 4x6 format is unable to print 8,5x11 format. Then, the composition will choose two labs, and will launch in parallel the two activities. Here, the horizontal constraints have an impact on the composition itself, and on the data flow.

B. Constraints modeling

In our model, we consider a community of services in which each service is of a given type. Types of services are defined by using unary predicates such as lab, delivery, bank, portal. For each service, we also distinguish several types of constraints, namely property constraints and QoS constraints.

Property constraints correspond to the static properties attached to a service. A service is defined with an input port, an output port, and a set of aspect constraints. For instance, a service of type photo lab can declare that it is able to deliver photos of size 4x6. This means that another service that will send to this service a set of JPEG images will receive photos of size 4x6. The size of the photos is a static property of the service.

QoS constraints depend on the operational behavior of a service, and is by nature, expressed with a set of dynamic constraints. These properties may vary given the operational context of the service. For instance, a photo lab service L1 will take 12 hours if the query arrives before 8PM, and 24 hours if the query arrives between 8PM and 6AM or if it arrives on saturday or sunday.

Constraints occur in input messages (as queries) and in output ones (as answers). For instance, when the client queries the 4x6 format, the data 4x6 belongs to the query. Similarly, when a photo lab service says it needs 24 hours to deliver the printed photos, the data 24 hours becomes a data connected to the message. The properties of a service X are defined by predicates of the form: propertyName(X, value [,value]). We can consider many predicates. In our example, maxDuration(X, t) is a predicate meaning that the service X will take at most t units of time to be executed. The predicate format(X, 4x6) is another predicate to define that the service X supports the 4x6 format.

Given the community of services, and their associated properties, we can define this community using a base of facts expressed as follows:

portal(p).
lab(l1), lab(l2), lab(l3).
delivery(d1), delivery(d2).
bank(b1).
format(l1,4x6), format(l2,4x6), format(l2,8x12).
maxDuration(l1,24h), maxDuration(l2,48h).
customer(l1,individual), customer(l2,individual).
customer(l3, professional)

Note that this base of facts does not contain any dynamic property. We consider that these dynamic properties can evolve during time and cannot be coded. The service can only raise the corresponding constraints when queried.

C. Abstract composition

The abstract composition defines only the way the different types of services are coordinated, i.e., the messages that will be exchanged between the different services, and their ordering. It can be considered as a skeleton, or a coordination pattern of a set of services. In our model, the flow can be defined using all the classical operators, i.e., sequence, parallel, choice, etc. ([9,10]).

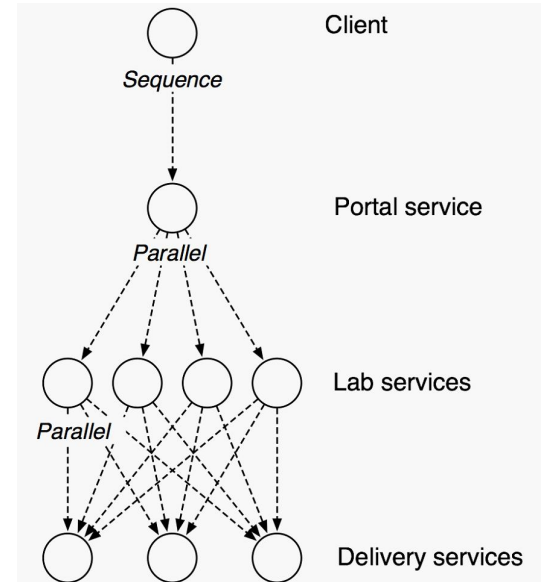Concerning our example, the abstract composition is as follows:

Figure 2: Abstract Composition

The portal knows that it will have a conversation with one or possibly several lab services (i.e. it has to send an appropriate message to at least one lab service), and a lab service knows that it has to send a message to a delivery service, and that before a lab finishes its work, it has to contact a bank in order to be paid. Thus, in the coordination pattern, we have all the conversations that have to be instantiated later.

Our main idea is that we can associate to each (composed) service S a constraint (program) specifying how S (here, the portal) makes use of its sub-services, and how to compute answers to queries sent to S. This constraint program, executed via a constraint solver, defines the composition of a service with its direct sub-services (here, the photolab services). Therefore, the composition is defined in an incremental way, since at each node of the composition, we define the connection with the successor nodes of the composition. It is important to note that solutions computed by sub-services are obtained dynamically by sending queries to their associated constraint programs, and by collecting solutions. Besides dynamic facts obtained via message passing, we also use the previous base of facts in which one can find different properties establishing the type of each concrete service and constraints connecting data and services.

In our model, we consider a binary predicate connect(X,Y) to specify that services X,Y can be connected. Moreover, we use a set of composition operators, like a sequential operator and a parallel operator parallel(X | C) meaning that all services X satisfying C are executed in parallel. For instance, the constraint cp=parallel(X | lab(X) ∧ connect(p,X)) allows us to execute in parallel all labs that can be connected to p. Note that it is possible to use other classical composition operators, e.g., choice, repeat, or if (guarded execution) [9,10].

Here, we can notice that the composition is still abstract since services are not yet instantiated (for instance, the variable X is not instantiated). To obtain an instantiation, we have to use a base of facts, stating a set of ground constraints. Back to our example, a possible base of facts could be:

portal(p).
lab(l1), lab(l2), lab(l3).
delivery(d1), delivery(d2).
bank(b1).
format(l1,4x6), format(l2,4x6), format(l2,8x12).
maxDuration(l1,24h), maxDuration(l2,48h).
customer(l1,individual), customer(l2,individual).
customers(l3, professional).
connect(p,l1), connect(p,l2).
connect(l1,b1), connect(l1,d1), connect(l1,d2).
connect(l2,b1), connect(l2,d2).

Given these facts, the solution of constraint cp executes in parallel l1 and l2.

The general principles of our approach can be summarized as follows: assume a query is sent to a composed service S. If it is an estimate query, then S asks direct sub-services for estimate queries in order to compute solutions for the client. Together with an estimate for the client, S returns a trace information describing the way concrete services are executed. This information is a constraint that can be processed by an execution query sent by the client to S. This constraint is used to model the workflow. The execution of service is monitored to check whether the constraint is satisfiable or not. When a problem occurs during execution, at least one constraint is violated. In that case, the monitor tries to change the execution process for the remaining tasks. A part of the whole constraint can be considered as ``rigid'' since it corresponds to tasks which cannot be undone. Given this rigid constraint, the idea is then to ask for a new estimate (an updated constraint) in order to get a new execution plan for the remaining tasks. Instantiation of abstract compositions and monitored execution of concrete compositions are discussed below.

D. Instantiating compositions

We present how an abstract composition can be instantiated to obtain a concrete one. Roughly speaking, this abstract composition is instantiated by analyzing a query that asks for an estimate. To illustrate the instantiation step, let us consider our example. Assume the portal gets the following query from the client: "Give me the price for printing 1000 photos in the 4x6 format, and I can get the printed result within 24 hours." Here, we have two constraints: the format constraint, and the temporal constraint. In order to fulfill the query, the portal will initiate a conversation with the instances of the lab services, in order to get the information necessary to return a price to the client. Following its constraint solver, the portal will only contact the services that are able to print the 4x6 format, in at least 1000 batches, and in less than 24 hours. So we may obtain for instance the following instantiation:
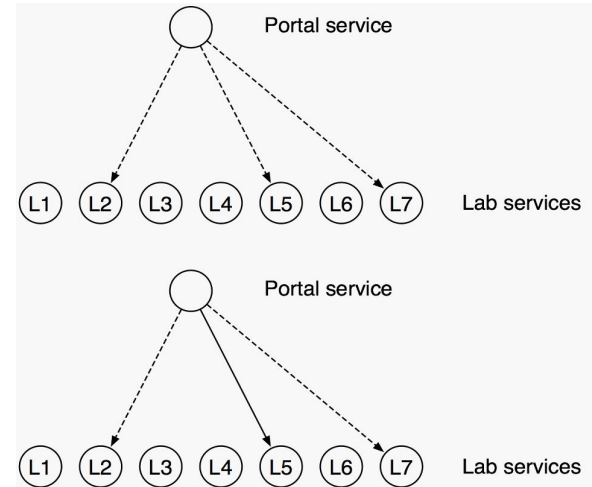

Figure 3: Instantiated and executable coordination patterns.

Figure 3 (left) says that only the labs L2, L5, and L7 are able to satisfied the constraints specified in the client query. The portal will then choose one of the available services. In Figure 3 (right), the portal chooses the lab service L5. The services L2 and L7 are no longer needed by the portal service. The flow of the composition has been set and the execution is now launched. As previously mentioned, the environment is naturally dynamic, and we also use the constraints to ensure the monitoring and the correct execution of the instantiation of the composite service. Let us explain how we deal with the monitoring and the real-time adaptation of the instantiation. Suppose the abstract composition given in Figure 4.a. This composition, determined by the vertical constraints, says that once the service L1 is finished, either the service D1 or the service D2 is started.
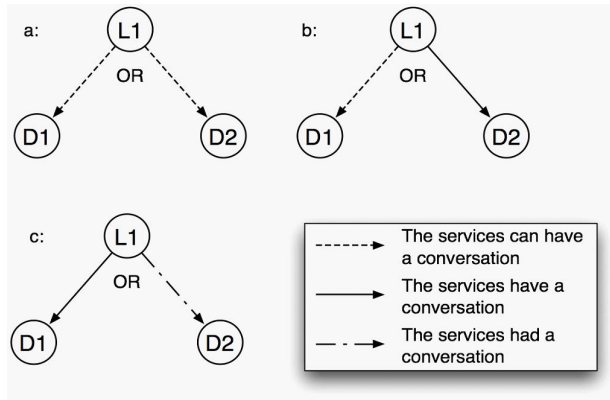
Figure 4: Dynamic selection of services.

The initial flow, computed with the horizontal constraints that are specified in the query, defines a conversation between the photo lab L1 and the delivery service D2 (Figure 4.b).

The instantiation step can be more generally described as follows. When a client initiates a query, the constraint solver associated to the service will first instantiate the abstract composition by solving constraints of the abstract composition together with the client query. The process is as follows:

1. the solver instantiates the constraint defining the composition, i.e., it finds some concrete services that fulfill the constraint,
2. the solver forwards the client query to selected concrete sub-services of the composition,
3. each sub-service computes solutions to their constraints,
4. finally, solutions are processed by the service, and the result is forwarded to the client.

In our example, a client sends a query estimate(number=1000, format=4x6, maxtime=24) which means that the client wants to print 1000 photos in the 4x6 format in less than 24 hours. The portal p receives the query, and it tries to solve this query in conjunction with cp, where cp defines the abstract composition constraint (using the parallel combinator of Section 3.3). At this stage of the reasoning process, we know that some photolabs satisfy the query. We proceed similarly for the instantiation of the delivery services.

To find services satisfying a given constraint, we proceed using a standard principle in constraint programming. Since a service comes with its properties, we check them in conjunction with the current constraint store. If this conjunction is satisfiable, the service can be selected, otherwise another service is tried. One can remark that constraints are first propagated to sub-services in a top-down manner; then solved constraints are propagated back in a bottom-up manner.

E. Executing and monitoring instantiated compositions

Once the abstract composition had been instantiated, the client is able to choose one of the possible instances according to its preferences. The composition is executed using the selected services. The composition is controlled by a monitor which checks the satisfiability of constraints during the execution process.

Using constraints is very powerful for monitoring the composition, and modifying it when something goes wrong. To illustrate the interest of monitoring, consider again our example. Assume that the client query is to have its photos in 24 hours. The instantiation process has selected the photolab that claims it will take 8 hours to print the photos and to send them to the delivery. The selected delivery service (D2) takes 16 hours to deliver the photos. Now, assume the photo lab service (L1) is late, and it will send the photos in 12 hours. In order to be sure that the horizontal constraints are still satisfied, the instantiation is adapted and the flow is modified and redirected to the delivery service D1 since it is faster than D2. Our model is flexible enough to handle these kinds of problems. In fact, the constraint store is updated with new constraints. Due to its estimate delivery of 12 hours, D1 becomes the new delivery service. Dynamically, the new flow is calculated using the horizontal constraints available on services (shown in Figure 4.c) without any impact on the client.

F. Handling cost constraints

In our framework, the idea is to use time-dependent cost functions to express different kinds of costs. For simplicity, we restrict ourselves to functions which are constants over some time intervals. Costs could be for instance the price of items or the (estimated) execution time of Web services. To each service S, we associate a cost function fS, which is constructed according to cost functions of sub-services S1,...,Sn, and how S composes its sub-services S1,...,Sn. For instance, if a service S calls two sub-services S1 and S2 sequentially, then costs are additive, and $fS(t)=fS1(t) + fS2(t)$. When the cost functions are interpreted as execution time, the execution time of a service S calling S1 and then S2 is $fS(t) = fS1(t) + fS2(t+fS1(t))$, which means that S2 is called at time $t+fS1(t)$ to determine the execution time. Note that if S1 and S2 are composed via a parallel construct, then the cost function of S is no longer additive.

To illustrate the computation of time cost functions, consider the following scenario. A photo lab service receives a query for 1000 printed pictures in 6x7 format. The service must give an answer to the query that includes how much time it will need to fulfill the work. This time will depend on its current workload, and on planned workload. The photo lab service will also query how much time a delivery service will need to deliver the photos to the client. It will include this time within its estimation. The time cost functions of a photo lab service and a delivery service can be expressed as functions which are constant over some intervals. By composing the photo lab service and the delivery, the execution time function is also given as a function which is constant over some intervals.

G   Optimization Problem

The selection of Web services, at each layer of the vertical composition, can be expressed as a constraint optimization problem. Given our example, several optimization criteria are possible:

- The portal sends all solutions to the client, who is in charge of choosing his preferred solution.
- The portal can automatically optimize one criterion, like the price or the estimated duration, in order to give an answer to a query such as ``I want an item x in less than 24 hours, with the cheapest price''.
- The portal is capable of performing multi-criteria optimizations. In that case, one could introduce (1) priorities on costs or (2) a balance on costs. For (1), we first optimize according to the criterion with the highest priority, it yields solutions which are optimized according to the remaining criteria. For (2), the portal has to optimize an objective function defined as a balanced sum of the different costs.

In all these cases, the constraint reasoning engine associated to a Web service is capable of computing optimized solutions when the query sent by the client involves an optimization problem.

IV. RELATED WORK

Web service composition is nowadays a very active research direction. Many approaches have been investigated including techniques based on planning in AI [1], situation calculus [2,3,4], conversational transition systems [5,11], or symbolic model-checking applied to planning [6]. Some other works are more related to the use of constraint solving techniques [12,13,14]. As in [12], we do not consider all the dimensions of the problem, since we assume that a pattern composition is already known, and we restrict ourselves to the problem of instantiating the variables of the pattern, i.e. the different kinds of Web services.

V. CONCLUSION

In this paper, we promote the use of constraint reasoning to implement a form of pattern instantiation. With respect to classical configuration problems, our approach has to cope with the dynamic aspect of Web services. We have proposed a framework where Web services interact via queries and answers, which are respectively the input and output messages exchanged by Web services. Hence, our approach relies on the analogy between the computation performed by a Web service and the execution of a constraint (logic) program. In order to take into account the dynamic behavior of Web services, the facts needed to execute the constraint reasoning engine of each Web service are not static, but are obtained dynamically by calling sub-services. Moreover, the process is monitored, to possibly change on-the-fly the current selection of Web services. This monitoring phase is also supported by the constraint engine associated to the Web service.

REFERENCES

[1]   Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, and Dana S. Nau, ``HTN planning for web service composition using shop2.,'' *J. Web Sem.*, vol. 1, no. 4, pp. 377-396, 2004.

[2]   Srini Narayanan and Sheila A. McIlraith, ``Simulation, verification and automated composition of web services,'' in *Proc. of WWW,* 2002, pp. 77-88.

[3]   Sheila A. McIlraith and Tran Cao Son, ``Adapting golog for composition of semantic web services.,'' in *Proc. of KR.* 2002, pp. 482-496, Morgan Kaufmann.

[4]   Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith, ``Planning with qualitative temporal preferences,'' in *Proc. of KR,* 2006, pp. 134-144.

[5]   Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su, ``Conversation specification: a new approach to design and analysis of e-service composition,'' in *Proc. of WWW,* 2003, pp. 403-410.

[6]   Marco Pistore, Annapaola Marconi, Piergiorgio Bertoli, and Paolo Traverso, ``Automated composition of web services by planning at the knowledge level.,'' in *IJCAI,* 2005, pp. 1252-1259.

[7]   Krzysztof R.Apt, *Principles of Constraint Programming,* Cambridge Univ. Press, 2003.

[8]   Rina Dechter, *Constraint Processing,* Morgan Kaufmann, 2003.

[9]   Eric Monfroy and Carlos Castro, ``A component language for hybrid solver cooperation,'' in *Proc. of ADVIS.* 2004, vol. 3261 of LNCS, pp. 192-202, Springer.

[10]   Tony Andrews et al., ``BPEL4WS: Business Process Execution Language for Web Services,'' Tech. Rep., July 2002.

[11]   Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella, ``Automatic composition of transition-based semantic web services with messaging,'' in *Proc. of VLDB.* 2005, pp. 613-624, ACM.

[12]   Ahlem Ben Hassine, Shigeo Matsubara, and Toru Ishida, ``A constraint-based approach to horizontal web service composition.,'' in *Proc. of Semantic Web Conference.* 2006, vol. 4273 of LNCS, pp. 130-143, Springer.

[13]   Nizamuddin Channa, Shanping Li, Abdul Wasim Shaikh, and Xiangjun Fu, ``Constraint satisfaction in dynamic web service composition.,'' in *Proc. of DEXA.* 2005, pp. 658-664, IEEE.

[14]   Rohit Aggarwal, Kunal Verma, John A. Miller, and William Milnor, ``Constraint driven web service composition in meteor-s.,'' in *Proc. of SCC.* 2004, pp. 23-30, IEEE.

Eric Monfroy. Is professor at the University of Nantes. He did a PhD on "Solver Collaboration for Constraint Logic Programming", defended in 1996. His research interests include decision support systems, cooperative constraint problem solving, constraint programming, constraint satisfaction and constrained optimization, constraint language design, hybrid constraint solving, rule based computation. He is currently on leave at UTFSM, Valparaíso, Chile.

Olivier Perrin. Is assistant professor at University Nancy 2 since 1995, and works at LORIA, a laboratory common to CNRS, INRIA and Universities of Nancy, in the ECOO project. His research interests include various topics in Virtual Organizations, Web Services Technologies, Workflow Management, and Enterprise Application. He has published several articles in international conferences and journals, and he was involved in many international and European projects. His current work is on extending Web services and Grid computing with constraints programming concepts.

Christophe Ringeissen. Is researcher at INRIA, and works at LORIA. He got his PhD in 1993 at the University of Nancy 1. The subject of his PhD was on the "Combination of constraint solving techniques". His research interests are automated deduction, constraint solving, semantics of declarative programming languages (algebraic and logia programming, constraint programming), and combination of constraint solvers.

Universidad Nacional de Colombia Sede Medellín
Facultad de Minas

Reseña Histórica

La Escuela Nacional de Minas fue fundada el 11 de abril de 1887, bajo la dirección del general Pedro Nel Ospina como rector y como Vice-rector Luís Tisnés, aunque el general Pedro Nel Ospina no se posesiono, elaboro con ayuda de su hermano Tulio los estatutos y reglamentos de la escuela, los cuales fueron una adaptación de los estatutos y reglamentos de la Escuela de Minas de California (Berkeley)los cuales fueron cambiando de acuerdo a las necesidades de cada década, en ellos se fomento una filosofía con valores cívicos, éticos y de orden por medio del estimulo y el ejemplo que comprometían el comportamiento del estudiante no solo dentro de la escuela sino fuera de ella, a demás se introdujeron hábitos de sobriedad, de economía y principios morales de honradez, honestidad y respeto.

En sus inicios contó con 22 alumnos matriculados, y luego d e tres meses fue cerrada por la poca cantidad de estudiantes, fue reabierta un año despué s, el 2 de enero de 1888, bajo la rectoría de Tulio Ospina V, esta vez contó con 27 alumnos matriculados y con un plan de estudios de 4 años de un mejor control de los pro gramas curriculares y adaptarlos a nuevas condiciones adelantándose a las necesidades futuras de la educación y asegurando así un buen desempeño de los futuros profesionales.

En 1906 la Escuela Nacional de Minas se anexo a la universi dad de Antioquia, a la que perteneció durante cinco años más, en 1911 paso a ser de nue vo una entidad independiente.

En 1940 la institución fue incorporada a la Universidad Nac ional y continúo con el nombre de Escuela Nacional de Minas, ese mismo año comenzó la co nstrucción de la actual sede, la cual fue inaugurada el 19 de diciembre de 1944, en el marco del primer Congreso Nacional de Ingenieros.

Entre 1941 y 1950 se crean las carreras de ingeniería geoló gica y petróleos y arquitectura, la cual se separo de la facultad de Minas en 1954, en1960 se crea la carrera de ingeniería administrativa, luego se crearon los progr amas de ingeniería industrial, ingeniería mecánica e ingeniería química y se separ aron los programas de ingeniería geológica y petróleos en dos programas diferentes, ac tualmente la Facultad de Minas Administra 11 programas de pregrado en ingeniería, 17 d e posgrado y cuatro doctorados.

La Facultad a lo largo de su existencia ha sido motora del desarrollo de la ciudad, del departamento y del país, a través de sus 12.000 egresados quiene s han constituido la mayor parte del personal dirigente y técnico en las explotacione s mineras, las construcciones de distinto tipo, la infraestructura vial, los de sarrollos hidroeléctricos, las obras de abastecimiento de agua, las obras sanitarias y la indus tria, así como en los planes de desarrollo físico, económico y social.