Applying Case-Based Learning to Improve the Efficiency in the Web Service Compositions

Ingrid-Durley Torres and Jaime Guzmán-Luna

Abstract—**Several planning techniques in artificial intelligence have been used to perform web service composition (semantic or not), but this process typically uses heuristics based planners combined with search techniques usually too expensive in time solution. In this article, we propose the use of case-based reasoning to reduce the computation times of composition; the model aims to infer from past experience a solution that would guide the selection process during a Web services composition. The proposed methodology also uses a classification defined by an algorithm of semantic similarity technique, in order to compare the new problem, with all previous problems. The previous problem with greatest similarity is accompanied by its corresponding solution and is used to specify which goals already achieved and what remain to achieve for the new problem. The result demonstrates greater efficiency, reducing the search space spending less time.**

Index Terms—**Composition of semantic web services, planning in artificial intelligence and case-based learning, INDYGO.**

I. Introduction

From the perspective of service composition, there are few approaches which integrate learning models to improve various aspects of the task of Web service. A particular model which proposes the use of planning in Web service composition is the concurrent planning and execution model (INDYGO) [1], [2], which Universidad Nacional at Medellin Campus designed and implemented, and aims to produce semantic web service compositions in real time while handling incomplete data. This model, like most service composers, carries out solutions without taking into account experiences of previous solutions. Due to this, it was deemed valuable to propose within the INDYGO model the implementation of case-based learning techniques to enable the improvement of its efficiency in matters concerning the composition process itself that the planner is in charge of. It is important to highlight that even though the model adjusts to a previous development, this proposal may also be easily adapted and implemented in other approaches which under AI planning techniques enable semantic Web service composition since it is in charge of formulating its solution relying on commonly known elements implemented within this dominion.

To amplify the proposal, this document is organized as follows: section two revises the reference framework related to the problem of service composition using AI planning techniques, specifically, centered on INDYGO. Section three, discusses and analyzes case-based learning concepts and proposes a representation of the solution starting from them; section four details the integration architecture of the CBR and INDYGO model and its functionality. Section five summarizes some results of the validation of the model, and section six presents conclusions and future work related to this proposal.

II. A Composition of Semantic Web Services

A. Semantic Web Services

A Semantic Web Service (SWS) is a Web service whose internal and external description is in a language that has interpretable semantics well-defined by machines [3].

With SWS, the aim is to: 1) Define exhaustive description models to describe Web services and its related aspects; 2) Support ontologies as the basic data model which enables machines to interpret data on the Web; 3) Define semantics-based technologies to automatize processes that use Web services. Currently, there are a series of languages which allow the specification of Semantic Web Services, highlighting: OWL-S [4].

OWL-S (*Ontology Web Language for Services*), defines an ontology and a language for SWS named OWL-S. Particularly, an ontology of an OWL ontology language and mechanisms related to it, allow the description of Semantic Web Services in terms of concepts and complex relations among them, including classes, subclass relations, and cardinality constraints, among others. In a more detailed form OWL-S, includes: 1) *ServiceProfile*: it focuses on what SWS does, since it describes necessary properties for SWS, for its automatic discovery, as its capabilities, its input and output, and preconditions and effects (possibly conditional). 2) *ServiceModel:* it focuses on how SWS works, since it describes an SWS process model, which is the control and flow of data involved in the use of SWS. It is designed to allow automatic composition and the execution of SWS. 3) El *ServiceGrounding*: it focuses on accessing SWS in terms of communication protocol and in the processes of serialization. This allows connecting a description model of communication protocol levels and the description of messages in WSDL [5].

B. INDYGO-SWS Composition

INDYGO proposes the application of techniques for web semantics and the planning of artificial intelligence in a web service composition model which faces ambiguity problems in the description of services and the handling of incomplete

Manuscript received June 16, 2013; revised August 7, 2013.

Ingrid-Durley Torres is with the Institución Salazar y Herrera from Medellin, Colombia (e-mail: i.torresp@iush.edu.co).

Jaime Guzmán-Luna is with the Director of SINTELWEB Research Group, Universidad Nacional de Colombia, Medellin Campus (e-mail: jaguzman@unal.edu.co).

DOI: 10.7763/IJET.2014.V6.702

information in the context of the Web. To deal with the aforesaid problems, the model allows the use of *OWL-S* services and implements a planning technique which handles the semantics of an open world in its reasoning process. As a result of this work, we obtained a web service composition system which includes: i) a module which interprets *OWL-S* services and converts them into a PDDL planning problem; ii) a planning module which handles the problem of incomplete knowledge and iii) a service execution module which interacts concurrently with the planner to execute each one of the services on the composition plan.

The planning algorithm, itself, is based on the idea of decomposing the original *PP* planning problem in *n* independent sub-problems. In which $PP = <F, L, A, S_o, G, m>$, where *F* is a set of numerical variables; *L* is a set letters or facts; *A* is a set of possible actions, S_o is the initial state of the problem, *G* is the set of objectives of the problem and *m* is the optimization criterion (metrics of the problem). So that it is subdivided in $PP_i =< F, L, A, S_o, G_i, m>$, $i=1....n$, where *n* corresponds to the number of objectives of the problem ($n=|G|$). Hence, a P_i plan, is different for each one of the sub problems. Finally, existing conflicts among said plans are studied to decide which action from which plan shall be the first to be executed (a_0). The planner sends a_0 to the executor when it requests it while the planner assumes the success of this operation and continues selecting the next action to execute (a_{next}); to expedite the process when the execution of a_0 finishes, the planner updates its internal representation of the world and gets ready to send (a_{next}) to executor just as long as the expected effect coincides with the real effect. Since state varies continuously during the execution of a plan, a term *S* (current state of the world) is introduced to refer to the model the planner has of the world at this moment. At the beginning of the planning process, the current state coincides with the initial problem ($S_o = S$). The new current state *S* is calculated as: *result* ($a_{next}, S_o, 0$); modification of the current state *S* takes place considering that the execution of a_{next} shall have the expected effects, that is, assuming there are no external factors ($\delta = 0$). This model of operation is called assumption-based planning [6]: actions are deemed determinists ($\delta = 0$); to simplify the real problem, and an alternative action is recalculated when the real effect of the actions does not coincide with the expected effect ($\delta \neq 0$). In case one of the actions fails during its execution, the executor informs the planner this failed event so that the planner may perform a new calculation for an alternative action while the executor waits, and after a respective time, it may ask the planner for the corresponding action.

All this process repeats continuously until the user decides to stop the execution of the planning agent or until all objectives are fulfilled and the planner returns a special *NOP* (no operation)-type action, which corresponds to an action without preconditions or objectives. In this case, it is considered a successful composition plan.

III. Case-Based Learning

A. Definition of Case-Based Learning

Case-based reasoning - CBR, is not more than just another problem-solution paradigm related to learning machines in AI [7], but it is precisely its differences with other AI approaches which make it so special. Instead of solely relying on general knowledge of the problem dominion, or carrying out actions throughout the relations among problem descriptions and conclusions, this paradigm is capable of using specific knowledge of previous experiences, in other words, situations of a concrete problem (cases).

Before the proposal of a problem not dealt with previously, there is an attempt to locate a previous similar case and adapt its solution to the situation of the new problem.

B. CBR Life Cycle

The classic case-based reasoning model [7], [8], divides a reasoning cycle into four stages: 1) Retrieving (*Retrieve*): aimed at finding and obtaining a previously saved case; the objective of this phase is to recover cases whose experience are potentially useful to resolve a new problem. This stage generally requires a combination of search and match techniques [9]; similarity measures are usually useful tools to find a case close to the search conducted. 2) Reusing (*Reuse*): this is the phase in charge of generating a new solution to the problem consulted starting from cases recovered in the previous stage [10]. 3) Revising (*Revise*): once the system has a solution adapted to a user's search, it is essential to assess how useful it is to solve the problem at hand; thus, revision work is based on an analysis of this solution within its own application framework in the real world. 4) Retaining (*Retain*): this phase is presented as the final phase of the CBR cycle in which the product obtained in the previous stages (solution of the problem) is included in the knowledge of the system. Even though there are various approaches related to this task, a widely accepted simplistic one is aimed at the storing of a new case which has been assigned to search the user's problem and the solution adapted by the system.

C. Methodology of Classification

As it was previously cited, a case must contain information both of the problem and of a respective solution. Thus, in general, we have a case which may be considered a record of a previous experience of a problem. Information stored regarding an experience depends both on dominion and the purpose for which the case is used. A description of a problem must include: 1) Goals that must be reached to resolve a problem. 2) Restrictions for these goals. 3) Characteristics of the situation of the problem and relations of parties. Other important information which must be stored includes the description of the solution which shall be used when we find ourselves in a similar situation.

The knowledge we need to resolve a specific problem is found grouped in a few cases or even in just one of them.

In tune with these considerations, and without detouring from the dominion of service composition, then you can say that a case may be defined as tuple:

$$c_i = \{dsc, ctx, prob, sol, hist\}$$

In which:

dsc: is a textual case description. *ctx*: represents a set of context-related characteristics of a case as role, case creator, etc. *prob*: is the definition of the composition problem defined by a composer's mechanism. *sol*: denotes a solution to composition problems and *hist*: historical case record.

Based on this formal definition and on the context of the knowledge schema presented in this topic, it is clear that the structure of a case for the proposed model must be a construction able to store information related to the dominion and the planning problem and the composition carried out for that problem in addition to context elements and a historical record of the context. In this sense, an *OWL-S* platform offers a suitable interphase for the representation of this knowledge. Hence: (**i**)solution of the case: it could be represented by a *Process Model* which describes a service that includes a previously–achieved composition solution, and a *Service Profile*, which allows the representation of both preconditions, inputs, outputs and effects related to a composed service. (**ii**) The problem: shall be represented by all OWL specifications, which associate the context of corresponding ontologies to participating services. Besides the user's requirement specification is expressed with a specification of a *Process Model* which must be fulfilled. Thus, the base of CBR knowledge shall be presented by a set of all previous successful correct solution compositions *of Process Models*. Such *Process* shall at the same time store a user's requirement, which shall be expressed as an objective state to reach a solution and the initial state which was defined to initiate the process.

Additionally, there is a type of reasoning implemented which is based on the transformational CBR cycle which is based on retrieving a case from a case database which most resembles the current problem. A search a user introduces is understood as a user's requirement; in other words, a *Process Model* which must be built with the objective you wish to fulfill, and lastly, cases which are also represented by the *Process Models* of previous solutions, which were valued by a similarity function. In this case, one to one of the *Process Models* of the cases of the case data base are compared with a user's until reaching a *Process Model* case which has the most elements in common with the requirement a user actually formulated. That retrieved case shall be adapted (if required), afterwards so that it may include all the elements expressed in a user's requirement (if all are not at least the possible maximums since there would be occasions in which it would not be possible for all of them to appear).

Since for each case, it is enough to store an index of previous solutions and their relation with a web address where the solution composition plan is found (composed service), at this point it is viable to think that the BDs are adequate technology for the task of retrieving known solutions of problems; even though it has limitations regarding the characterization of the dominion, the advantages are sufficiently in accordance with the representation of the composition problem you wish to conduct as a case. Furthermore, considering that the characteristics which shall be represented are not at all numerous (an Id and a web service URL).

Regarding a knowledge representation technique, a sequential flat memory search technique has been chosen, even though the recovery of cases is made slow and does not always act as the most expensive in reference to other representation techniques; moreover, new solution storage comes out cheaper since it only depends on its addition via the identification of an index.

A case retrieval process has decided to incorporate, two techniques tending to filter most exactly the degree of similarity of a new problem with previously stored case solutions. The first of these techniques is semantics, which using specifications immersed in the description of (OWL-s) services enables expediting the recovery process itself preselecting just those cases which coincide with the exact semantic specification of the problem. In addition to a semantic technique, we have decided to include another simpler technique, which via a mathematical similarity function, makes it possible to evaluate from a previous selection the degree of similarity of that case, in which the sum of all the weights of the characteristics adjust at a higher value with the new case. Hence, the possible number of instantiated cases which coincide with the new problem are extensively reduced rendering a result equal to a highest degree of coincidence.

Finally, a case adaptation process is found; in this aspect, given the multiplicity of these elements in a composition process, represented by the type of data and the value of the same parameter both for input and *output* elements of their own services, it has been decided to implement a parameter adjustment technique which may initially identify that the parameters of the new problem coincide with the retrieved case. Once the different elements have been identified, you must resort to a transformational technique which enriches the previous solution with the new elements now generated. Thus, you avoid the recalculation of complete plans, when there are partially instantiated solutions.

IV. Proposed Architecture

Then, a developed CBR follows a classic processing cycle (Retrieving, Reusing, and Validating Learning) as the one shown in Fig. 1. Hence, once a search has been formulated, expressed in the specification of a respective dominion (web services, enunciated in OWL-s), it is processed as follows:

A. *Case Retrieve Module*

It consists in selecting from the knowledge database those cases whose description most adjusts to the information presented in the new case. To do so, a semantic similarity measure has been chosen as a retrieval algorithm. The way that retrieval measure works is cited as follows.

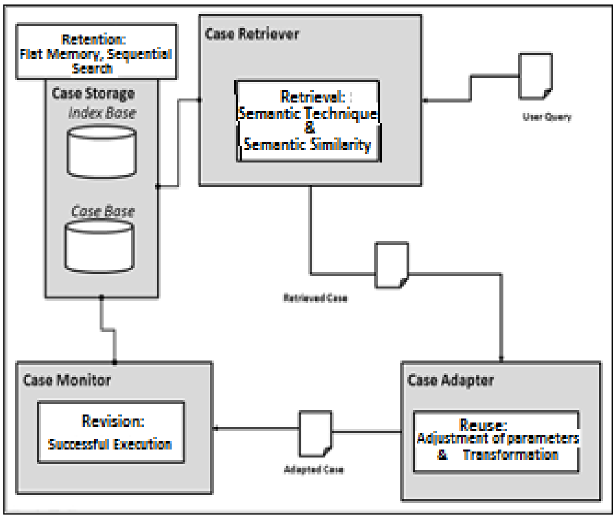


Fig. 1. CBR model architecture

1) Semantic Technique Algorithm: a semantic algorithm is directly related to the ontological characterization of previously stored composition plans. The specification of the superior Web service ontology proposed herein has been retaken from INDYGO [4] and allows the characterization of service functionality in terms of service input and output parameters along with their preconditions and effects. To do so, we have used the characterization of an instance of a service via a *profile:Profile* class, which describes a service in terms of its input (*process:Input*), outputs (*process:Output*), preconditions (*expr:Condition*) and results (*process:Result*).

Inputs and outputs are defined in terms of parameter specification (that is a *process:Parameter* class). This type of parameter is also a subclass of a *process:ProcessVar* class which refers to a process variable which conducts the service. The latter class is composed of two fields: a parameter type (*process:parameterType*) and a parameter value (*process:parameterValue*), which are respectively any URI type (*xsd:anyURI*) and literal XML type (*rdf:XMLLiteral*).

A precondition is represented in terms of a Condition class of an ontology Expression (*expr:Condition*). Since this Condition class agrees with an OWL-S specification, it is a subclass of Expression of the ontology Expression (*expr:Expresion*), which is composed by the following fields: an Expression language (*expr:LogicLanguage*) which allows the specification of the URI reference (*xsd:anyURI*) where you can find the specification of the logic language that shall be used in the expression of a condition. The body of expression (*expr:expresionBody*) which enables the description the body itself of the expression associated to a condition.

Results (*process:Result*) have effects represented in terms of an Expression class defined in an Expression ontology (*expr:Expresion*), which has previously been explained.

On the whole, with the above conceptualization, it is possible to perceive that: a dominion may be expressed as an *OD* ontological specification. Likewise, the problem shall be expressed as a conjunction: of an ontological specification of an initial state (*OEI*) and an ontological specification of a target state (*OGS*). With this representation, now, it is possible to represent the following algorithm.

2) Similarity technique algorithm: First, a new composition problem named *NPNC*, is composed by a dominion specification (OD_{NPNC}), an initial state (OEI_{NPNC}) and a target state (OGS_{NPNC}). Such a problem must be compared with a previously resolved problem and also stored under the name of $NPPC_i$ (where *i* corresponds to a solution storage index) also defined by a dominion specification (OD_{NPPC}), an initial state (OEI_{NPPC}) and a target state (OGS_{NPPC}). Thus, this way you compare:

$$NPNC(OEI_{NPNC}) <--> NPPC_i(OEI_{NPPC}) \tag{1}$$

$$NPNC(OGS_{NPNC}) <--> NPPC_i\ (OGS_{NPPC}) \tag{2}$$

To do so, you evaluate the semantic similarity measure defined as:

$$SimSem^{OEI} = \frac{OEI_{NPNC} \cap OEI_{NPPC}}{OEI_{NPNC} \cup OEI_{NPPC}} \wedge SimSem^{OGS} = \frac{OGS_{NPNC} \cap OGS_{NPPC}}{OGS_{NPNC} \cup OGS_{NPPC}} \tag{3}$$

Then :

$$SimSem^{Total} = \frac{SimSem^{OEI} + SimSem^{OGS}}{2} \tag{4}$$

This process is carried out for each previously stored solution, in other words since:

$$NPPC_{(i=0....n)} \rightarrow simSem_i^{Total} = \frac{SimSem_i^{OEI} + SimSem_i^{OGS}}{2} \tag{5}$$

and for each one, you conduct a comparison process, generating a $SimSem_i^{Total}$ equation for each *i* comparison. Each value is orderly stored in a temporary file so that once all the previously stored problems are compared, you may measure semantic similarity having the greatest value to be retrieved and later adapted.

B. Case Storage or Storage Module

A case storage module works with two specific databases: 1) the first is named *Index Base*, which is in charge of storing an index which acts as a sole identifier assigned by the system for each stored case and the web address aims at the *Process Model* which defines a previously resolved compound web service. The problem of indexation is an answer to a data accessibility problem, and it refers to a correct selection of the most important attributes as case structure and organization, and the retrieval indexes of the most important cases. An index of a case library is in its most elemental implementation a pointer towards a case. Then, accessibility shall depend on the combination of tasks and the current situation, and on the correct selection of attributes which distinguish one case from another marking something important of a case which distinguishes it from the rest. In this work that attribute shall correspond to the abstract ontology of a *Process Model* which represents a compound web service composition. In other words, an index within an *Index Case Base* is generated for each abstract ontology 2) The second database, named *Case Base*, defines contents as such for each *Process Model*, in other words, instantiated ontologies belonging to each case. These shall be associated to an index of one of previously stored abstract ontologies. It is important to consider that there shall be no instance which is not associated to one of the *Index* indexes.

Cases are stored sequentially in a simple list, a file, which shall be transferred to a persistent storage in a database. This technique is commonly known as sequential flat-memory search. Thus, every time you wish to store a case, it shall be added after the last index previously recorded. While to retrieve a case, it shall be necessary to work sequentially (that is, one to one and in a given order) each one of the cases previously stored.

C. Case Monitor (Monitor of Cases)

Before storing a case, the system must monitor that it effectively is associated to a successful execution. To do so, the model has included an event listener module, in charge of evaluating the correct execution of the composition plan, specifically known as a valid plan. A valid *P* plan (*valid_plan (P, I, G, {δ_i}))*, is that which is successful when executed on an initial state *I*, reaching a *G* target state.

$$valid\_plan(P,I,G,\{\delta_i\})=\mathcal{V} \leftrightarrow result(P,I,\{\delta_i\})=S' \wedge goalstate(S', G) = \mathcal{V} \tag{6}$$

Sequence *$\{\delta_i\}$ equals* $\{\delta_0, \delta_1, \ldots., \delta_n\}$ and defines a succession of unforeseen situations that have taken place during the execution of the plan. This sequence makes that the execution of the same plan to resolve the same problem does not always render the same result.

D. Case Adapter (or Case Adapter Module)

A case adapter module initiates, once it is determined, a case which most resembles a current problem to adopt it to adjust to peculiarities. That is to say, the solution proposed via a retrieved case is sent to the planner module to replan the context of a composition. The functionality of this model acts under the following principles: 1) Previous problem descriptors are compared with a new problem's descriptors, and differences are drawn from them. 2) Specialized adjustment heuristics are applied to a previous solution to create a new solution. Those aforesaid heuristics shall capture relations between problem characteristics and solution parameters.

In this case, a composition process cycle repeats for different elements, in accordance with the structure of a retrieved case, passing through a Case Monitor Module.

Once the success of the execution assessed in the *Case Monitor* has been compared, a previous solution is enriched and adjusted with new parameters, and finally, it is stored as a new case in the case database (*Storage Case*).

Once you know how the composer system works, the following step is to design a model which may integrate the exposed CBR model with the INDYGO model previously detailed.

V. Integration Model

An integrated model was named JABY and as detailed in Fig. 2, this architecture is divided into three large modules; one is represented by a CBR model, another corresponds to web service composition architecture, adapted from INDYGO, and lastly, an integrator wrapper module in charge of carrying out respective conversions in the specification language of each one of the first two modules. At the same time, each module is subdivided into a set of elements belonging to its functionality.

The functional schema of the module functions as follows: once a user has deployed an application, the user must specify that user's system composition requirement, expressed in three associated parameters: 1) a description of an initial state represented by a URL of the initial ontology, described in OWL; 2) a description of a target state represented by a URL of a final ontology, also expressed in OWL; 3) The supply of a *.txt-extension file, with a list of the URL of the OWL-S of services recorded in the repository and which potentially participate in a composition plan. After the parameters are received, they are transferred to a retrieval module, which is in charge of identifying and extracting objects found in the definition of an initial state with the purpose to conduct a search for relevant cases for a set of objects (instances). The above was done to reduce the number of cases to evaluate semantically; and thus, save and expedite calculations in latter stages of retrieval. Once the lists of relevant cases have been obtained for each instance of an initial state of the definition of a user's search, they are intersected in such a way that you obtain a list in which each case belonging to it is relevant for all instances of the initial state a user defines.

This case list is sent to a semantic evaluator, which calculates for each case the semantic similarity value between a user's search and *serviceProfile* definitions of a stored case.

Using the values of the semantic similarity value, the system selects the case with the largest semantic similarity value; if there happens to be more than one case with an equal semantic similarity value, the system shall select the first of a sequential order.

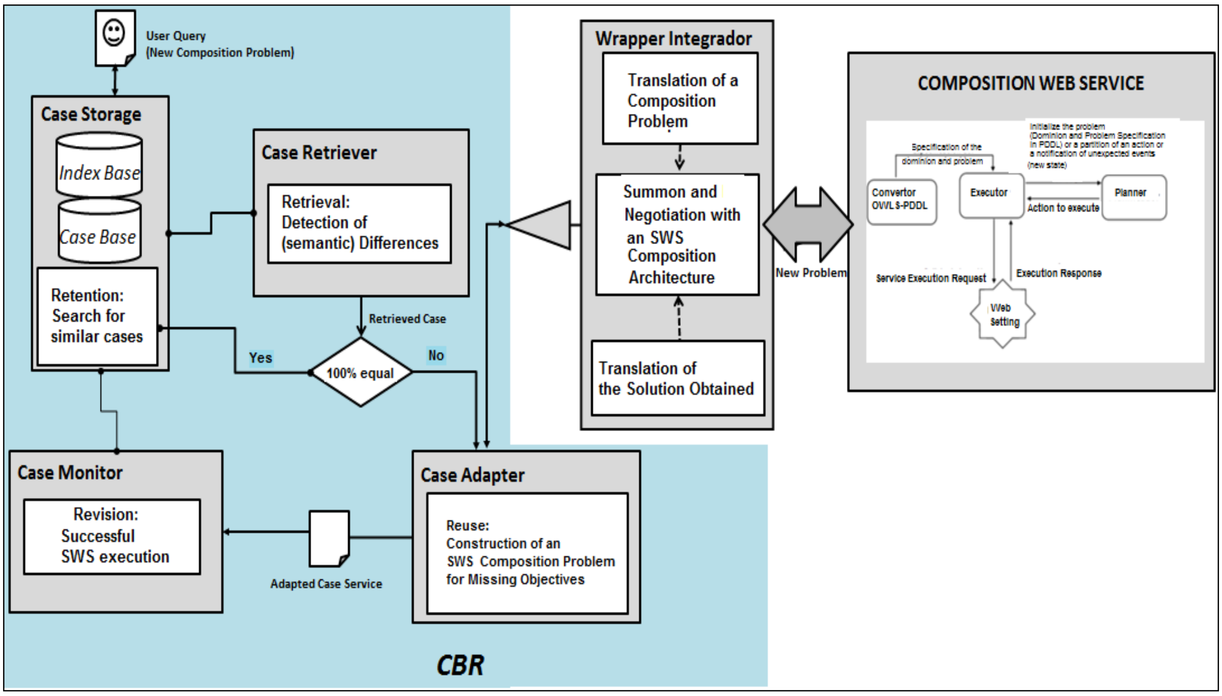


Fig. 2. CBR- INDYGO integration model

A chosen case is sent to an adapter module along with input parameters initially provided by a user; then, the adapter component conducts an evaluation of the completeness of the case proposed related to its objectives. That is to say, a user's objectives are not fulfilled by executing a selected case when a list of missing objectives resulting from this evaluation is empty; an adaptation process was avoided and the selected case is sent directly to a monitor module. On the other hand, a new final ontology is constructed using the missing objectives with which along with a service list and a definition of an initial state, the architecture of semantic web service is invoked to obtain a sequence of semantic web services, in such a way that they may allow the complementation of remaining objectives. At the same time, this sequence is delivered to the system as an OWL process Model which is integrated to a partial solution retrieved from a selected case.

Finally, the solution plan which integrates a partially retrieved solution and an adaptation carried out along with a composition architecture is sent to a monitor module which for each specification of the anatomic process (*atomic process* de OWL) within the solution plan locates a WSDL descriptor related to the inside of *grounding* and constructs an SOAP message to invoke a service. The response to the SOAP request a posteriori to the service execution is validated by the monitor to evaluate the usefulness of the case.

Once the execution process of each one of the actions that make up the plan have concluded their invocation and a user's request has been fulfilled satisfactorily, the module stores it in the case database and the indexes of the process composed which contain a sequence and a list of atomic services which represent the *process model* of an executed composed service, as well as the description of the composition request then user delivered.

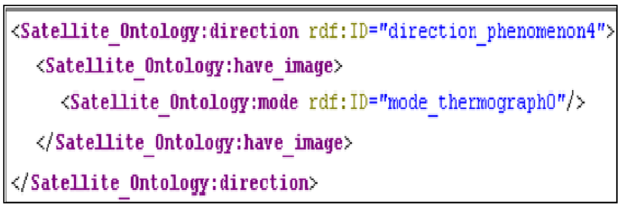
VI. A Case Study

To evaluate the proposed prototype, a series of tests were conducted with the aim to compare the efficiency and scalability of reaching JABY goals, with a basic INDYGO composer. To do so, several versions of the satellite dominion were constructed. The cases the CBR stored were more complex than the ones formulated by a user in that user's search. The following is a description of objectives associated to a user's search, (see Fig. 3 and Fig. 4), and objectives associated to cases to be retrieved in which case they make a partial contribution of 1 and 2 objectives.

Table I compares INDYGO and JABY processing times for the various levels of completeness of a retrieved cases (1, 2) objectives.

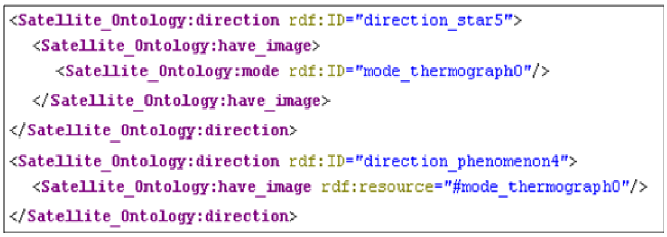
TABLE I. Scalability of Satellite Dominion Objectives

N° of Operations	INDYGO	JABY CASO 1 (GOAL 1)	JABY CASO 2 (GOAL 2)
5	303,575	144,547	50,946
50	306,299	151,047	76,345
75	306,893	161,89	84,3
100	366,510	187,215	102,369
250	315,509	215,828	137,584
500	328,184	247,459	182,752



```
<Satellite_Ontology:direction rdf:ID="direction_phenomenon4">
  <Satellite_Ontology:have_image>
    <Satellite_Ontology:mode rdf:ID="mode_thermograph0"/>
  </Satellite_Ontology:have_image>
</Satellite_Ontology:direction>
```

Fig. 3. Case 1- Objective representation diagram



```
<Satellite_Ontology:direction rdf:ID="direction_star5">
  <Satellite_Ontology:have_image>
    <Satellite_Ontology:mode rdf:ID="mode_thermograph0"/>
  </Satellite_Ontology:have_image>
</Satellite_Ontology:direction>
<Satellite_Ontology:direction rdf:ID="direction_phenomenon4">
  <Satellite_Ontology:have_image rdf:resource="#mode_thermograph0"/>
</Satellite_Ontology:direction>
```

Fig. 4. Case 2- Objective representation diagram

Fig. 5 shows the relation that there is between the completeness of a problem and INDYGO and JABY processing time. Although the computer cost that comes along with the adaptation of a case, turns out to be high in comparison with the processing time consumed by INDYGO in the construction of a solution starting from scratch, it is 33% less for the first case and 50% less for the second case. This is why it is more beneficial.

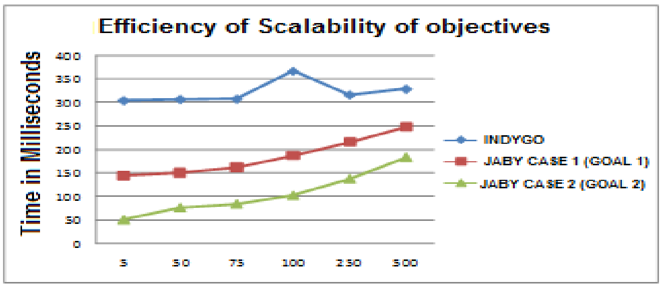


Fig. 5. Objective completeness diagram

VII. Conclusions

In this work, we start from the fact that conceptually intelligent agents and (semantic) web services were conceived with totally uneven purposes, and as such, it is understood (as a hypothesis) that they must remain in two different levels of abstraction. The main idea which substantiates the technology of agents is not that intelligent agents are able to provide services, but that they are conceived as autonomous entities which incorporate intelligence and cognitive capabilities which enable them to show pro-active behavior aimed at objectives and to establish interaction processes, either competitive or cooperative, with other entities to fulfill their design objectives. Ontologies as such are components which enable establishing communication between agents and Web services located at various levels of abstraction fluently and without erroneous interpretations. Another valuable element is constituted by a heuristic classification model which combines the three technologies (agents, ontologies and services), with (heuristic) AI techniques to allow the exploitation of dominion semantics and services resulting in a process of automatic SWS classification.

ACKNOWLEDGMENT

This paper presents preliminary results of the project “apoyo al grupo de sistemas inteligentes web-sintelweb” con código quipú 205010011129, developed by the universidad nacional de colombia, in agreement with the gea group of institución universitaria salazar y herrera.

REFERENCES

[1] J. Guzmán and A. Ovalle, “Web services planning agent in dynamic environments with incomplete information and time restrictions,” in *Proc. The 11th IEEE Internacional Conference 2008, Computational Science and Engineering Worksho*p, pp. 245-250, 2008.

[2] J. Guzmán and D. Ovalle, “Web service composition: a semantic web and automated planning technique application,” *Revista de Ingeniería e Investigación*, vol. 28, no. 3, pp. 145-149, Dec.2008.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services. Concepts, Architectures and Applications*, Springer, 2004.

[4] M. López, D. Mscherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. Cox, and K. Forbus, “Retrieval, reuse, revision and retention in cased based reasoning,” *The Knowledge Engieneering Review*, Cambridge University Ed. Press., vol. 20-3, pp 215-240, 2006.

[5] WSDL., Web Service Semantics. April 2005. [Online]. Available: http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.pdf.

[6] Koenig S. Tovey, C. A. Smirnov, and Y. V., “Performance bounds for planning in unknown terrain,” *Springer-Verlang*, vol. 1- 2, no. 147, pp. 253-279, 2003.

[7] A. Aamodt and E. Plaza, “Case-based reasoning: Foundational issues, methodological variations, and system approaches,” *Computer Science and Artificial Intelligence*, vol. 7, no. 1, pp 39-54, 1994.

[8] A. Bregón, M. Aránzazu, J. Rodríguez, C. Alonso, B. Pulido, and I. Moro, “Early fault classification in dynamic systems using case-based reasoning,” *Lecture Notes in Computer Science,* vol. 4177, pp. 211-220, 2006.

[9] B. Limthanmaphon and Y. Zhang, “Web Service Composition with Case-Based Reasoning, Database Technologies, Performance bounds for planning in unknown terrain,” in *Proceedings of the 14th Australasian Database Conference (ADC2003)*, Australia, February 2003. Pp-201-208.

[10] OWL Services Coalition. OWL-S: Semantic markup for web services, 2006 [Online]. OWL-S White Paper. Available: http://www.daml.org/services/owl-s/0.9/owl-s.pdf.

Ingrid-Durley Torres received her Ms.C. degree from the Faculty of system engineering, Universidad Nacional de Colombia, Medellin Campus. Where actually is Ph.D Student and working as teacher research at Institución Salazar y Herrera from Medellin, Colombia**.** Her topic investigation is Artificial Intelligence (planning, semantic web, e-learning).

Jaime Guzman-Luna received B.Sc.in civil engineering, and then received his Ms.C and Ph.D. degrees in system engineering from the Universidad Nacional de Colombia, Medellin Campus. Where actually is Director of SINTELWEB research group and working as teacher Universidad Nacional de Colombia, Medellin Campus, Colombia**.** His topic investigation are Artificial Intelligence (planning, semantic web, web services and robotic).