# Implementing an composition architecture for an online game software

# Implementando una arquitectura de composición para un software de juego en línea

*Jaime Alberto Guzmán Luna*[*], *Ingrid Durley Torres Pardo, Arlex David Martínez*

SINTELWEB research group, Computer Science Department. National University of Colombia. Carrera 80 No 65 - 223 Bloque M8A. Medellín, Colombia.

(Recibido el 2 de diciembre de 2010. Aceptado el 10 de noviembre de 2011)

## Abstract

This paper presents architecture for Web service composition, in which a plan of composition is constructed based on an agent planning, which can be executed in a concurrent way during its composition in order to estimate the following action to be executed instead of preparing a complete plan which frequently would be invalidated. This feature is invaluable when it comes to addressing problems in real time. Specifically we propose a test domain online games software, called ENVIRO.

----- *Keywords:* Web architecture for Web service composition, agent planning, problems in real time and online game software ENVIRO

## Resumen

Este artículo presenta una arquitectura para la composición de servicios Web, en la que un plan de composición se construye sobre la base de un agente de planificación, que puede ser ejecutado en forma concurrente durante su composición a fin de estimar la siguiente acción a ejecutar en lugar de preparar un completo plan que con frecuencia será invalidado. Esta característica es muy valiosa cuando se trata de abordar los problemas en tiempo real. Específicamente se propone como un dominio de prueba un software de juegos en línea, llamado ENVIRO.

----- *Palabras clave:* Una arquitectura de Composición de Servicios Web, agente de planificación, problemas de tiempo real y software de Juego en línea ENVIRO

* Autor de correspondencia: teléfono: + 57 + 4 + 425 53 78, correo electrónico: jaguzman@unal.edu.co (J. Guzmán)

## Introduction

Web services composition has been widely studied during the last years. One of the main approaches to carry out this work comes from Artificial Intelligence (AI) planning [1]. Today, there are different tools available, among which we can stress on [2, 3]. However, although these applications work appropriately, it has been identified that many of them are not capable to jointly face several problems related to Web context, such as: (i) It cannot be expected to have all relevant information on the system local knowledge base; for that reason, the planner, when having incomplete information, will need to collect some information with the purpose of solving composition problem; (ii) For most users there are limitations on the Web in relation to the time the system can use to deliberate before acting; and (iii) Web is a highly dynamic world; then, an effect produced by a service is not always known or predictable. In order to face jointly problems mentioned above, this document presents a service composition architecture which allows focusing on a planning agent which allows estimation of the following action to be executed instead of preparing a complete plan which frequently would be invalidated by the environment dynamism. Under this approach, planner is able to obtain a response (service to be executed) in a limited interval of time and handling the incomplete information related to the Web. This feature is invaluable when it comes to addressing problems in real time, specifically we propose as a test domain environment for games software, which is run from a finite set of web services, previously stored in a repository. Such services will be implemented in a plan of composition, from which you can run online, a custom action game (gathering resources). Decide what action to build an entire change of plan then must be repaired, is a feature that reduces the time of composition almost instantaneously responding to the requirements of a user player.

This paper is organized in the following way: The following paragraph describes the architecture composition. Then presents the modeling languages used by the model. Of course is illustrates the planning agent. Later, describes game software as the test environment. Then is evaluates the functionality of the architecture. Finally, presents the conclusions.

## Composition architecture

Proposed composition architecture is based on the use of on-line planning techniques, which bear in mind the environment concept. In our case, the environment describes world status on the Web, which can be observed and modified through actions executed by a set of agents. In our approach, the concept of agent [4], it we used to refer to any phenomenon capable to alter the world. An external agent is that phenomenon which can act on the environment on which planning agent does not have any kind of control. This lack of control as well as the big diversity of external agents make that study of these agents becomes not very useful. However, changes produced by external agents in the Web environment are one of the main causes of execution failures of services associated to plan actions that planning agent detects when executing a plan.

Proposed planning agent's characteristics are based on already existing proposals, such as TCA (Task Control Architecture) [5], 3T Three-tired architecture [6] and Simplanner [7], which follow the following design principles: (i) concurrence: Several processes, such as environment monitoring, execution, and planning are carried out in a concurrent way; (ii) Reactivity of the system is favored by an architecture organized by levels, in which highest levels show a more complex behavior and represent information with a higher abstraction level.

Planning agent architecture, consists of three main modules (see figure 1): (i) A translator of OWL-S [8] specification to XSPDDL[9], which translates initial domain and goal state ontologies, together with service descriptions respectively implemented in OWL [10] and OWL-S, in a domain specification and its corresponding planning problem in XSPDDL; (ii) a planner,

which according to this knowledge, tries to find out which the best services (represented by actions) to be executed are, in order to reach objectives; (iii) an executor, which takes service descriptions under a planning environment and translates them into execution orders which can be understood by existing actual Web services. This executor is communicating with a service repository specifically designed for the test environment. Once success has been validated in a service execution adds the results to their facts base.

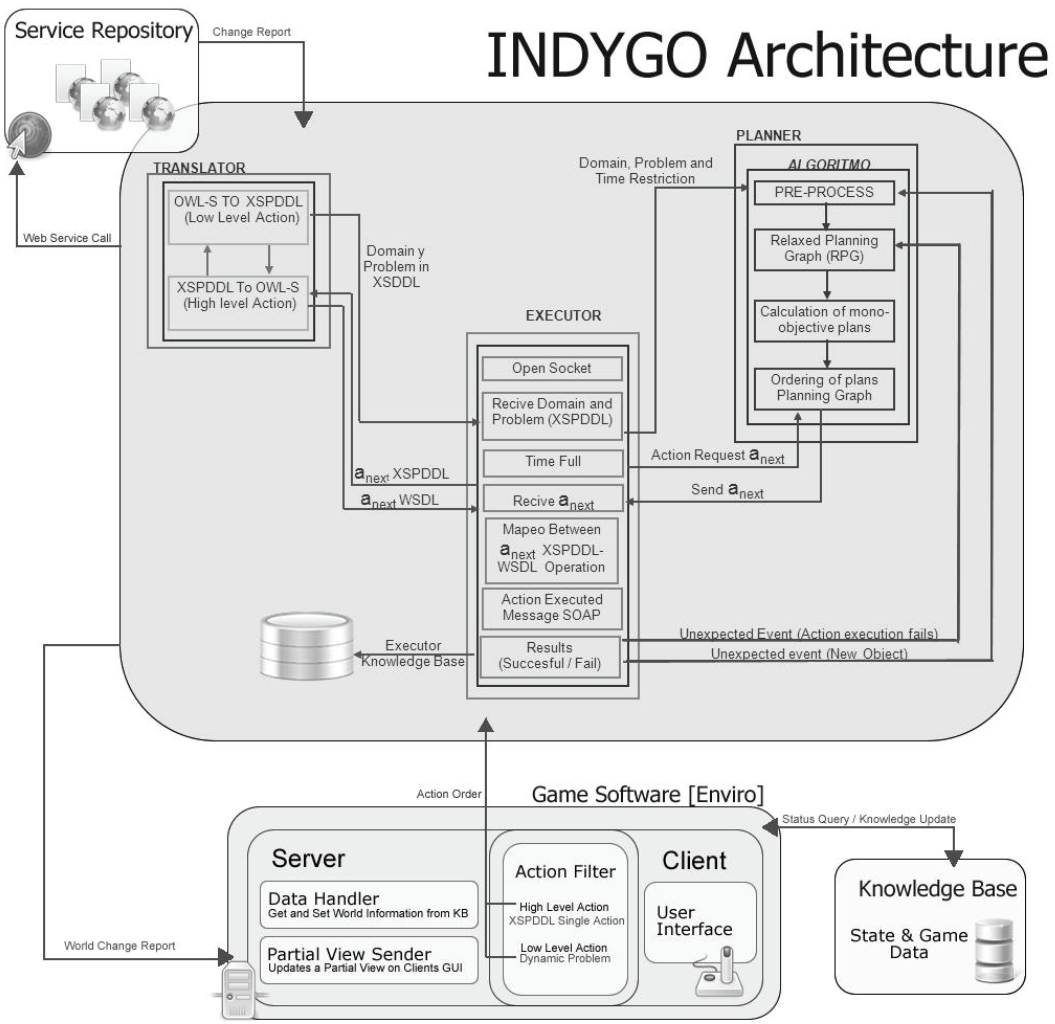


**Figure 1** Planning agent architecture

The integration of the planning agent architecture is carried out through the use of test environment, named ENVIRO. The planning agent more ENVIRO formed the INDYGO architecture. ENVIRO, software game, is a game whose features are represented in Semantic Web Services (SWS). The state of the game is in the area or unit objective and detailed description of the status of units, buildings and items specific to the game map (trees, civil buildings, solder etc.) this description is taken from the information embedded in the server via the command executed from the GUI for a client. In developing the game we consider two types of actions: (i) shares directly running

the SWS and (ii) composite actions require a plan of composition to achieve the objective. This software automatically transforms the objective of the player (user) in a composition problem defined by OWL-S ontologies.

## Modeling languages

In the proposed architecture, one can distinguish two basic languages to represent a composition domain and services: OWL-s and XSPDDL.

### *Services with ontologies*

Today, there are several proposals to implement Semantic Web Services (SWS) [11]. Among them, we can mention OWL-S, WSMO / WSML [12], and WSDL-S [13]. In our architecture, it takes a set of available OWL-S services, a domain description consisting of OWL ontology and a planning query as input. In the specific case of the description of the domain, the ontology used contains all types and instances that represent values that make up the domain. Furthermore, it contains the properties and their instances which represent the state it is in when beginning the composition.

In the case of the planning query, the agent takes an ontology that resembles the domain but it differs from the ontology of the description of the domain in that it only contains instances of the properties that describe the conditions that the domain must have at the end of the composition.

In order to handle the problem of incomplete information, when modeling OWL-S services, we have included the concepts of effects which change the state of the world and effects which change the agent's knowledge.

The first enable us to represent the change of state in the Web World by executing a service and the latter enable us to represent the agent's mental changes when executing a service without altering the Web World. For this reason, in the effects of SWS we use labels <*knowledge-effect*> and <*effect*> for identifying expressions that represent effects of knowledge and effects of change the world. These expressions for reasons of convenience have been implemented using XSPDDL, for which we have extended the OWL-S to distinguish the use of such expression.

### *XSPDDL world modeling*

As most classical planners, the planning module of agent needs a description of both domain and problem through a modeling language. For that purpose, we defined XSPDDL, we have defined a XSPDDL, which is an XML-Schema to which can be written in XML, planning problems PDDL 1.2 specification [14].

In order to represent the incomplete information, a tri-value logic has been used about the set of literals which define a state. If planning agent knows the truth value of a literal, such literal should explicitly appear on the state specification. If such literal does not meet such state, it should be preceded by the clause not. On the other hand, if agent does not know a specific fact, it should not appear on the status specification. Proposed representation, however, is compact, thus avoiding making the tedious task of listing all unknown facts on a state. To model the effects of world change and the effects knowledge change in the SWS, we have chosen to implement two types of actions related to one same service within the planning component. The first is called *Action*, which represents a service with world changing effects. The second is called *Sensing*, which represents a service with knowledge changing effects. This avoids making large changes to the PDDL syntax. With the purpose of modeling the dynamic environment of the Web in our planner because PDDL does not allow representing non deterministic actions (actions which applied on a same state, can produce different results), our agent's planning mechanism XSPDDL has the following features: (i) Planner does not explicitly manage indeterminism for it, all actions are considered deterministic actions, for this reason, planner only works with expected effects in its actions. (ii) Indeterminism is handled during the execution: executor verifies the real result

of each service execution through monitoring mechanism. If a non-expected result is found, planner repairs the plan to adapt it to the new situation.

### *Translating OWL ontologies to XSPDDL*

Web service ontologies, initial ontology of domain and goal ontology of planning query are translated to a domain and a problem under AI planning approach. This requires transferring specifications of ontologies to XSPDDL. The Class and Properties included in OWL ontologies are mapped to Types and Predicates of XSPDDL. Web services are mapped to XSPDDL Actions, in which the model of an action represents a Web service. Thus the main relations of conversion among OWL-S in XSPDDL are summarized in figure 2.

```
<actions>
- <action name="op_1_recollect_resource">
  - <parameters>
      <param type="Unit">?UnitData</param>
      <param type="Source">?Resource</param>
      <param type="Position">?XPos</param>
      <param type="Position">?YPos</param>
    </parameters>
  - <precondition>
    - <and>
      - <and>
        - <pred name="agentKnows">
            <param>?Resource</param>
          </pred>
        - <pred name="agentKnows">
            <param>?Unit</param>
          </pred>
        - <pred name="agentKnows">
            <param>?XPos</param>
          </pred>
        - <pred name="agentKnows">
            <param>?YPos</param>
          </pred>
        - <pred name="Object_in_XPosition">
            <param>?Resource</param>
            <param>?XPos</param>
          </pred>
        - <pred name="Object_in_YPosition">
            <param>?Resource</param>
            <param>?YPos</param>
          </pred>
        - <pred name="Object_in_XPosition">
            <param>?Unit</param>
            <param>?XPos</param>
          </pred>
        - <pred name="Object_in_YPosition">
            <param>?Unit</param>
            <param>?YPos</param>
          </pred>
        </and>
      </and>
    </precondition>
  </action>
</actions>
```

**Figure 2** OWL-S to XSPDDL Conversion

## Planning agent

A planning problem $P = (O; I; G)$ is a triplet where $O$ is the group of operators, $I$ is the initial state, and $G$ as the goals to be accomplished. To solve this, in the context of the tree problems to be solved (dynamic environment of the Web, incomplete information and time restrictions) proposed planner's algorithm is provided with four steps proposed planner's algorithm is provided with four steps: the Pre-process, relaxed planning graph, generation of Mono-objective plans and ordering of plans. At the end of the last step of proposed algorithm, the following action to be executed is selected among all of the actions which compose each one of mono-objective plans. Then, specification of such actions is sent to the executor for its corresponding execution. This process is repeated continuously until the user decides to stop the planner's execution or if the planner achieves all objectives.

### *Pre-process*

In this step, information about the problem and domain, specified through XSPDDL in files generated by OWL-S to XSPDDL converter, is processed and organized. This step is divided in three tasks: (i) verification of domain and XSPDDL problem through a parser; (ii) analysis of possibilities for reaching objectives, in which all possible actions which lead to a satisfaction state are generated, (iii) data structure to store all the previous information.

### *Relaxed Planning Graph (RPG)*

The RPGs, provides necessary heuristic information for the construction of plans [15]. In our agent, we have modified traditional RPG, because we should have in mind the partial knowledge of Web.

In our agent's RPG, due to the tri-valued logics it uses, literal levels will be called propositional levels because these will have logic propositions (not literal ones). The first level $L_0$, will have all logic propositions which are satisfied in $S_0$.

Actions levels $A_t$ have all actions, which positive and negative pre-conditions are found in level $L_t$. The following level $L_{t+1}$, extends $L_t$ with positive and negative effects of actions $A_t$. A propositional level can have two propositions which represent the same literal, one in a positive way and the other one in a negative way. In this way, in t steps of time, we can obtain that a $L_t$ will then have $P_i$ and $\neg P_i$ propositions. The main difference with traditional RPG is that the RPG expands according to cost all possible actions until accomplishing the objectives (see figure 3 lines 15 - 36).

```
1.   Second expansión RPG until reaching all objectives
2.     while ∃ g ∈ G / cost (g) = ∞ do
3.           // Cálculo de la lista de acciones de sensorización
4.           L sens = { a ∈ A_t / ∃ l_i ∈ Leff (α) ∧ (cost ( l_i ) = ∞ ∧ cost ( ¬ L_i) = ∞ ) }
5.           if L_sens = 0, then fail endif
6.           // Acción de sensorización de menor costo
7.           α = argmín (cost_reach (α_i) + cost (α_i ) ), ∀ α_i ∈ L_sens
8.           // Cálculo de los nuevos efectos que produce a en el RPG
9.           New_Eff = { l_i } U { l_i }, ∀ l_i ∈ Leff (α) / cost ( l_i ) = ∞ ∧ cost (¬ l_i) = ∞ )
10.          // Inserción de los efectos de α
11.          t = cost_reach (α) + cost (α)
12.          cost (p) = t, ∀ p ∈ New_Eff, k ≥ 1
13.          L_k = L_k U New_Eff, k ≥ 1
14.    New_Prop = 0
15.    //Expansion RPG until reaching all objectives
16.      while ∃ g ∈ G / g ∈L_t do
17.           // Niveles de acciones
18.           A_t = { a ∈ A / Lprec (a) ∈ L_t) U { a ∈ A / Lprec (a) ∈ L_t
19.           cost_reach (a) = Σ p ∈ Lprec (a) Cost (p) ∀ a ∈ A_t
20.           // Costo de los efectos de las acciones de At
21.           for all p_i ∈ Leff (a), a ∈ A_t do
22.                 new_cost = mín (cost_reach (a_i) + cost (a_i)), ∀ a_i ∈ A_t / p_i ∈ Leff (a_i)
23.                cost (pi) = mín (cost (p_i), new_cost)
24.           end for
25.           // Nuevas proposiciones alcanzadas
26.           New_Prop = New_Prop U Leff (a) - L_t ∀ a ∈ A_t
27.           if New_Prop = 0, then fail endif
28.           // Siguiente nivel proposicional
29.           next_t = mín (cost (p) ) ∀ p ∈ New_Prop
30.           Lnext_t = L_t U { p ∈ New_Prop / cost (p) = next_t }
31.           New_Prop = New_Prop - Lnext_t
32.           t = next_t
33.      endwhile
34.           call expansión RPG2
35.    endwhile
```

**Figure 3** RPG algorithm with sensing actions Effects

One of the main objectives of this proposal is to work with incomplete information about environment, so sensing mechanisms are required. To accomplish this, we distinguish normal actions ($a$) from sensing actions ($\alpha$) and modify the RPG.

Consequently, if RPG is expanding, sensing actions in the graph are included, but not adding its effects in propositional levels. If in this first expansion, all propositional objectives of the problem are achieved, then, it is not necessary to acquire new information. If this is not the case, it is necessary to include possible effects of sensing actions in the graph. In order to accomplish this, a second expansion of RPG is necessary using algorithm in figure 3 (lines 1 – 14). If after this process, any objective of the problem becomes unreachable, it does not have a solution.

### Calculation of mono-objective plans

The agent planning uses objectives decomposition technique, where the algorithm calculates, in a concurrent way, a $P_i$ plan, separately, for each one of the propositional objectives of the problem ($g_i \in G$). The calculation of each $P_i$ plan is carried out in an incremental way: an initial plan (possibly incomplete) is constructed and is refined with time. This allows the interruption of the process at any moment and gives the planner an anytime behavior. The $P_i$ plan refining ends when a valid plan is reached, supposing that there are not unexpected situations, ($\delta = 0$). $P_i$ plans are constructed in a regressive way, not bearing in mind numerical pre-conditions of the actions. Initial plans guaranty that the first action is executable in the current status ($exec(first(P_i), S_0) = V$), this allows that these actions can be a possible answer from the planner.

Many times, a selected $P_i$ plan, may result not valid when being executed due to: (i) there is an action (Web service), which does not work ($a_{fail}$) because it has propositional pre-conditions which are not satisfied and (ii) there is an action which dose not work ($a_{fail}$) which only has not-satisfied numerical pre-conditions. For all this, after each $P_i$ is calculated, there comes a refining phase which is carried out while the executor does not requires an action, reports an unexpected situation or reaches a valid plan.

### Ordering of plans

In this stage, it is possible to find several $P_i$ plans, which are not totally executable, for each one of $g_i$ objectives. When the executor requires an action, the refining stage is stop. The action which returns to the executor is the first action of one or more of the $P_i$ plans. To reach this, we establish order relations between plans $P_i \rightarrow P_j$, in which $P_i$ and $P_j$, are two mono-objective plans. After applying the respective processes that evaluate the previous criteria, the planner chooses one of the plans which has already been ordered. The first action $a_{next}$ of the chosen plan is sent to the executor. The planner must then update its believes with the effects of $a_{next}$, and then recalculates $a_{next}$, which the planner then sends to the executor when it requests it once more.

### Game software

The distributed architecture of the test environment, software game "ENVIRO" has as its basic components: (i) a knowledge base of the domain itself, (ii) a server (iii) a filter action, and (iv) an interaction module with the user (see figure 4).

In the knowledge base stores all the information that is necessary for the development of the game (location of units, buildings, troops and resources). The server is responsible for managing all the information in the knowledge base, including changes to the outturn of the game commands. The filter action is responsible for filtering the direct orders, under the command of the game; of indirect commands, which correspond to complex objectives (gathering resources, repairing tanks, buy game software) that require a plan composition. And finally, a module of user interaction and core of the testing environment, which provides behavior and tools for searching information about the world and the corresponding game state display presented to the player (Data Handler + GUI) . When a player gives a hint, the software automatically generates a problem of composition, as defined in ontologies (initial and final).

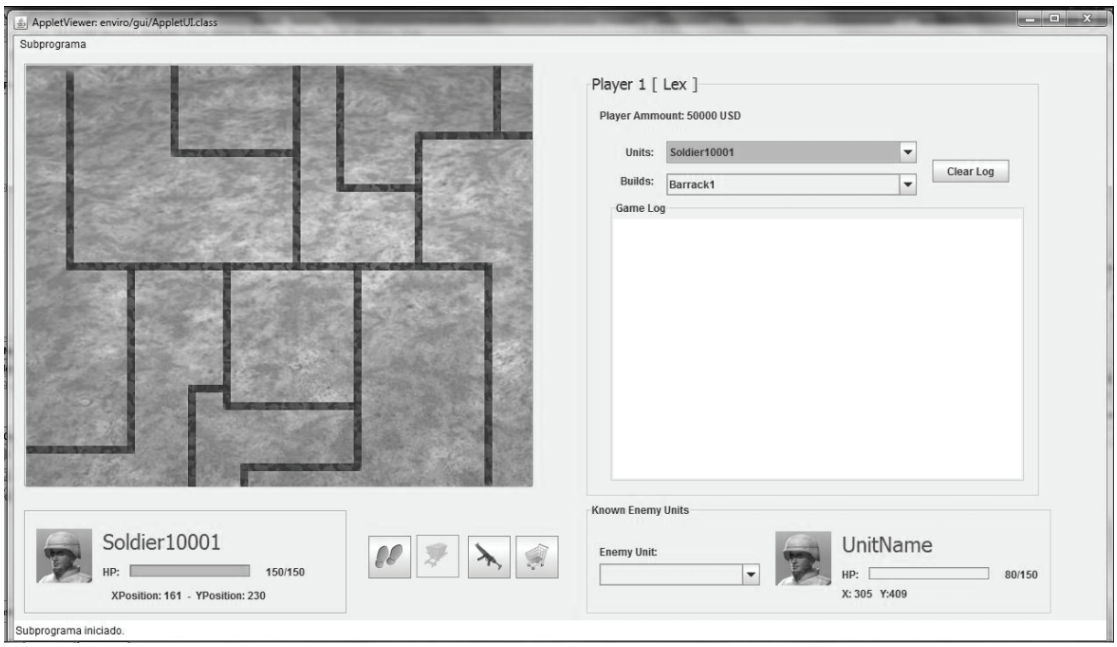


**Figure 4** Enviro GUI

This problem is reported to the planning agent through its executor, to be solved. After a period of time within the architecture, ENVIRO reviews and compares the facts as the performer with its base of knowledge, such a comparison is done periodically until the composition plan actions executed one by one, are been implemented in its entirety. If as a result of this comparison are different facts, ENVIRO updates its knowledge base, to continue the development of the game. This process is performed many times as the player determines his goals.

## Results

The purpose of these experiments is to assess the efficiency of our approach, i.e., the possibility of generating translations, parsings, instancings and executions in the smallest possible time. We ran our experiments on an Intel Pentium D 940 (3.2GHz Dual-Core), OCZ DDR2 2GB 800MHz Dual-Channel, JVM 1.6.0_13 64 bit, and under a Windows operating system.

The typical problems that planning aims to address and that are used in the international planning competition are so called closed problems in which the number of objects in the world remains constant. In ENVIRO game, however, creating objects is key. A typical game starts with a small number of units and a limited amount of resources. Then, those units can create structures that can produce new units, mine resources, or perform other functions. But one important aspect of this game is the buy "objects", for this specific case the user's problem required purchased through the Shopping Domain Service a weapon to advance in the game. This allows us to generate an idea of the efficiency of our proposal. Remember that this software automatically transforms the Objectives of the player (user) in a composition problem defined by OWL-S ontologies (figure 5). The example domain described here is intended for the purchase of software a weapon (*items*) through Web services (*ShopingDomain*) and consists of three Web services: (i) to determine if the item exists for this vendor (*In Catalogue Services*) (ii) obtain detailed information about the item (*GetIteminformation*) and (iii) to purchase the Item (*BuyItem*). Importantly, if the cumulative score in the game towards the player was not high enough, the weapon was likely not available in the catalog of

the seller. Otherwise, the catalog would show the entire set of items, weapons, shields and other items according to the player's score were available for purchase by the player. The first assessment is to list various service providers to purchase are recorded in our game.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:protege="http://pro
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://67.223.239.230/indygo/Domains/Shopping/General/Shopping_GoalOntology.ov
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:Shopping_Ontology="http://67.223.239.230/indygo/Domains/Shopping/General/Shoppin
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://67.223.239.230/indygo/Domains/Shopping/General/Shopping_GoalOntology
- <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://67.223.239.230/indygo/Domains/Shopping/General/Shopp
  </owl:Ontology>
  <Shopping_Ontology:Item rdf:ID="Item_12303" />
- <Shopping_Ontology:Client rdf:ID="Client_client">
    <Shopping_Ontology:Owner_possess rdf:resource="#Item_12303" />
  </Shopping_Ontology:Client>
</rdf:RDF>
<!-- Created with Protege (with OWL Plugin 3.2, Build 355)  http://protege.stanford.edu
```

**Figure 5** Composition problem defined by OWL-S shopping ontology

Now it will consider the objective described in table 1, there are states that "the customer owns the item 12303," where the item 12303, corresponds to the coding of the weapon. The case has been solved in four different ways, which are listed in table 1. The cases are different instances of the situation described above, with differences in the availability of the product with item 12303. In case number 1, the item is available in the store service_004, but not in the store service_001, in case 2, the item is only available to buy in the store service_001 in case 3, the item is not available in any store and if number 4, both stores have availability of the item.

Table 1 lists the first column on the number case; columns 2 and 3 indicate which stores have availability. Column 4 lists the number of web services implemented under the plan. Column 5 indicates the composition plan metrics achieved, column 6 records the length of the plan until they achieved the objectives or until it notifies the scheduler itself unreachable. Finally, column 7, records each of the services has been implemented within the plan, indicating which of them the feedback has coincided with a "*false*", typified by the executor as a kind of *Fail*.

By way of example shows first the case 2, for which INDYGO, hopes to buy in the store service004, however when running the respective *getiteminformation_service004T* information service, which investigates the availability of the item turns out to be false, which fail it notifies the planning mechanism, who immediately receives the report, looking for a new alternative plan from the current state of knowledge (planning agent), enabling the customer to meet the precondition to purchase the item 12303. Therefore, included in the new plan, the services provided by the store service001, indicating that the service *getiteminformation* first run, which confirms the correctness of its effects, allowing you to continue with the rest of the plan to research the goal. The example of case 3 corresponds to the full implementation of all possible services from both stores that achieve the goal.

Although the table does not allow registering the full name of the service instances executed (due to space limitations), the scheme never fails to be satisfactory, as expected since none of the two stores offer product availability (see columns 2 and 3 of table 1). However, in all cases, the information services have been

executed successfully, but his answer always correspond to false, INDYGO, shows that correctly follows the search for alternatives, whose instances match the data of the objective. Although apparently runs several times the service op_3_getiteminformation004service_12303, this corresponds to different instances of the same example: 1) _TitleData_build1 and 2) _TitleData_ soldier. Finally, when the user (player) executed all plan

**Table 1** Execution the action for buy item (weapon)

| *Case* | *Domain 1. Service004* | *Domain 2. Service001* | *No. of WS executed* | *Metrica* | *Length at plan* | *Plan* |
|---|---|---|---|---|---|---|
| 1 | Item available | Item not available | 1 | 3 | 2 | 1.op_3_getiteminformation_services004_item12303<br>2.op_11_buyitem_services004_item12303 |
| 2 | Item not available | Item available | 2 | 3 | 3 | 1.op_3_getiteminformation_services004_item12303 (FAIL)<br>2. .op_3_getiteminformation_services001_item12303<br>2.op_14_buyitem_services001_item12303 |
| 3 | Item not available | Item not available | 21 | 16 | 23 | 1.op_3_getiteminformation_services004_item12303 (FAIL)<br>2. op_9_getiteminformation_services001_item12303 (FAIL)<br>2.op_5_register_services004_item12303(FAIL)<br>3. op_11_register_services001_item12303(FAIL)<br>4. op_3_getiteminformation_services004_item12303 (FAIL)<br>5....<br>24. op_3_getiteminformation_services004_item12303 (FAIL) |
| 4 | Item available | Item available | 1 | 3 | 2 | 1.op_3_getiteminformation_services004_item12303<br>2.op_11_buyitem_services004_item12303 |

Another important aspect to consider when dealing with real problems and online game software it's represented in the time. The table 2, records the time in seconds of CPU consumed in each process of INDYGO, for exactly the same four cases in table 1.

**Table 2** Time of composer mechanism

| *No. Case* | *No. SW of information executed* | *Translation Time (msg)* | *Planning Time (msg)* | *Execution Time (msg)* | *Concurrent planning and execution* |
|---|---|---|---|---|---|
| 1 | 1 | 2.340 | 0.031 | 0.171 | 9.456 |
| 2 | 2 | 2.215 | 0.047 | 0.218 | 13.025 |
| 3 | 21 | 2.293 | 0.047 | 0.390 | 119.070 |
| 4 | 1 | 2.340 | 0.031 | 0.156 | 9.885 |

The final results are shown in the figure 6. As mentioned above, when information of availability of the item turns out to be false, the scheduler has to discard the plan currently being followed and calculate a new one to repeatedly select another $a_{next}$ possibly to assess the availability to run true, and these correspond $a_{next}$ to different instances of the same service. The cost of treating this type of "unexpected events" associated with incomplete information it nevertheless represents a very low increase in the planning process, since the values of time planning the curve tend to be constant.

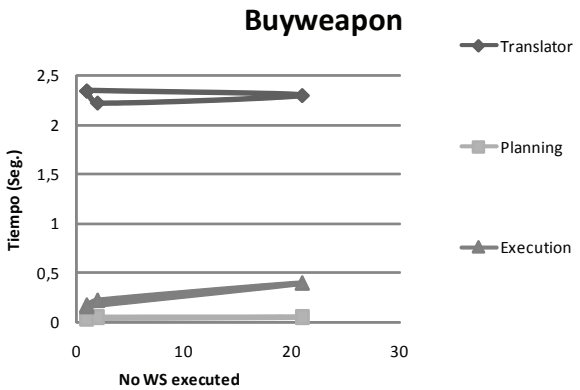


**Figure 6** Composition time

## Conclusions

An architecture for Web service composition has been proposed, which allows Web service composition described in OWL-S through the use of an OWL-S to XSPDDL translator, and planning agent which is based on a heuristic planning algorithm, designed to work in dynamic environments in a concurrent way with the executor. Planning agent includes some characteristics, not very common ones, for working in real domains. Since it supports sensing actions, it carries out planning with time restrictions; it uses number functions, it bears metrics in mind in order to optimize the problem and handles uncertainty. The basic idea is to accomplish plans, for each objective level separately. The test environment is a software game named ENVIRO that allows simulated several game behaviors with the composite SWS, which has shown its great possibilities for this kind of problems. The architecture and integration of ENVIRO with the planner agent have been described in the paper.

As future work, we propose to extend the expressiveness of agent using quantification operators to handle composition queries more complex. Another feature that we intend to improve in the agent is to minimize the fails in the resolution of a composition.

## Ackowledgements

This work is partially funded by the Doctoral Thesis "Multi-Agent Model of Concurrent Planning and Execution for the Semantic Web Services Composition in Partially Observable environments".

## References

1. M. Ghallab, D. Nau, P. Traverso. *Automated Task Planning. Theory and Practice*. Ed. Morgan Kaufmann. Trento (Italy). 2004. pp.449 -461

2. OWL-XPlan online: http://projects.semWebcentral.org/projects/owls-xplan/. Accessed in November 2010.

3. J. Peer. *A POP-based Replaning Agent for Automatic Web Service Composition. The Semantic Web: Research and Applications.* Second European Semantic Web Conference, Heraklion. Crete, Greece. May 29 - June 1, 2005. pp. 1-14.

4. E. Sirin, D. Martin, B. Mark, M. Drew McDermott, S. McIlraith, P. Massimo, S. Katia, M. Deborah, S. Naveen Srinivasa. "Bringing Semantics to Web services with OWL-S". *World Wide Web Journal Special Issue on Web Services: Theory and Practice*. 2006. pp. 26-40.

5. C. Pelachaud, J. C. Karpoutzis, D. Pelé, Martin, G. Chollet. *Intelligent Virtual Agents.* 7th International Conference. IVA 2007. Paris, France. September 17-19, 2007. pp. 47-53.

6. R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. Miller, M. Slack. "Experiences with an Architecture for Intelligent, Reactive Agents". *Journal of Experimental and Theoretical Artificial Intelligence*. Vol. 9(1). 1997. pp. 237-256.

7. O. Sapena, E. Onaindía. "Handling Numeric Criteria in Relaxed Planning Graphs". *Advances in Artificial Intelligence*. IBERAMIA. 2004. pp 114-123.

8. OWL-S Coalition: OWL Web Services 1.1. Online: http://www.daml.org/services/owls Accessed in November 2010.

9. XSPDDL. Online: http://67.223.239.230/xspddl. Accessed in Octuber 2010.

10. M. Dean, G. Schreiber, eds. *OWL Web Ontology Language Reference, W3C Recommendation*. Online: http://www.w3.org/TR/owl-ref/. Accessed in february 2004.

11. C. Bussler, A. Maedche, D. Fensel. *Conceptual Architecture for Semantic Web Enabled Web Services.* Memories Proceedings of the First International Semantic Web Conference on the Semantic Web. June 9 de 2002. pp 12-18

12. WSML. Working Group: Web service Modeling. Online: Language. http://www.wsmo.org/wsml/. Accessed in November 2010.

13. WSDL-S. Web Service Semantics WSDL-S, online: http://lsdis.cs.uga.edu/library/download/WSDL-SV1.pdf. Accessed in Octuber 2010.

14. PDDL: Planning Domain Definition Language online: http://www.ida.liu.se/~TDDA13/labbar/planning/2003/writing.html. Accessed in June 2010.

15. S. J. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach.* 2nd ed. Ed. Pearson Education. Spain. 2004 pp 37-42.