Obligation-based Agent Conversations for Semantic Web Service Composition

J. Octavio Gutiérrez-García
CINVESTAV del IPN – Unidad Guadalajara &
Grenoble Institute of Technology – LCIS
jgutier@gdl.cinvestav.mx

Félix F. Ramos-Corchado
CINVESTAV del IPN – Unidad Guadalajara
Av. Científica 1145, Col. El Bajío, Zapopan 45010,
Jalisco, México
framos@gdl.cinvestav.mx

Jean-Luc Koning
Grenoble Institute of Technology – LCIS
50 rue Laffemas - BP 54, 26902 Valence cedex 9, France
jean-luc.koning@grenoble-inp.fr

Abstract—We deal with composition of semantic web services to which access is controlled by agents. We propose a conversation-based web service composition method. We conceive web services as actions with preconditions and effects, expressed in terms of social norms, particularly obligations. We argue that the inclusion of obligation-based agents' conversations aids to lead the composition of services. In order to achieve this, we introduce an agent communication language that defines how messages affect agents' state, and thus, the access to their services. We also propose a method to automatically create a generic composer agent that is able to manage and compose web services, by means of inducing obligations to their participants.

Semantic web service composition; multi-agent systems; agent communication languages; social norms

I. INTRODUCTION

In this paper, we deal with dynamic composition of Semantic Web Services. Semantic web services emerge from the synergy of semantic web and web services. The semantic web attempts to convert the data stored in the web into knowledge that can be interpreted by computers, as well as humans [3]. The knowledge structure of the semantic web is provided by means of ontologies, where shared knowledge is represented by making use of classes, relations and functions [9]. This eases the discovery, selection and composition of semantic web services.

The necessity of efficient and effective web service composition methods arises when complex user requirements have to be fulfilled, and there is not a single web service that could satisfy such requirements. However, there could be the case that a specific combination of web services can satisfy the user requirements. The cost of achieving such combination manually is expensive, slow, and error-prone, therefore, automatic approaches to semantic web service composition are required, in order to fully exploit the semantic web (see [5] for a survey of semantic web service composition methods).

We follow the direction given by W3C, where it is stipulated that a web service should be implemented by an agent [4]. We provide an agent infrastructure to support the obligation-based interaction [10]. In addition, we define web services as actions, with their corresponding preconditions and effects, which are at the same time, defined in terms of obligations.

Moreover, we provide a high level agent communication language (ACL), which provides the operational semantics of the interaction. This ACL is based on speech act theory [15] similar to the researches presented in [6], [8], and [18]. The illocutionary acts are defined by means of operations over agents' obligations.

We propose a conversation-based approach for semantic web service composition. We design an algorithm to create a generic composer agent, which manages the interaction among web services, then, we translate the composition service process to a conversation planning process.

This paper is structured as follows: section II defines obligations and the operations that affect them; section III includes a definition of web services in terms of obligations, as well as the agent's infrastructure to support such web services; section IV delineates speech acts with obligation semantics; section V explains the proposed composition method based on agents' conversations; section VI shows an illustrative example of use of our approach; section VII presents close related work; finally, section VIII presents some concluding remarks and the future work.

II. THE OBLIGATION APPROACH

We define obligations as impositions to oneself to reach a state of affairs, demanded by a social force [10]. Our notion of obligation is akin to the one presented in [1].

An agent adopts obligations with the aim of exchanging them for rights to induce obligations to other agents. The symbolic representation of an obligation is given by $O(a, f)$, which means that an agent $a$ is obliged to reach the state of affairs $f$. Obligations can express prohibitions as obligations to not reach a state of affairs, represented by $O(a, \neg f)$. The states of an obligation are: allocated and released.

As an extension of simple obligations, we have conditional obligations. An agent is obliged to reach a state of affairs, whenever a certain condition holds. This is represented by $CO(a, f, g)$, where $g$ represents the necessary
condition. We make use of conditional obligations to implement external induction of obligations. Where agent \( a \) is instantiated by the receptor of the obligation and the fluent \( g \) represents agent \( a \)’s decision of accepting the fluent \( f \) as obligation (more details will be explained in the next sections). A conditional obligation is active only if the state of affairs \( g \) is held while the obligation is allocated. Hence, the states of a conditional obligation are: allocated-active, allocated-inactive, and released.

In order to formalize obligations and their management, we have selected the Event Calculus (EC) formalism, for its intuitive definition of actions.

**A. The Event Calculus Formalism**

The EC is a temporal formalism that provides a structure to reason about events and the time when these occur [12]. An EC predicate can contain three kinds of elements: an action \( act() \), a fluent \( f \) that represents a property that is affected by the action at certain point of time \( t \). We use the EC presented in [17]. The EC predicates utilized are:

- \( \text{Initiates}(act(), f, t) \): means that the fluent \( f \) holds after the execution of \( act() \) at time \( t \).
- \( \text{Terminates}(act(), f, t) \): denotes that the fluent \( f \) does not hold after the execution of the action \( act() \) at time \( t \).
- \( \text{HoldsAt}(f, t) \): means that the fluent \( f \) holds at time \( t \).
- \( \text{Happens}(act(), t) \): means that the action \( act() \) is executed at time \( t \).
- \( \text{InitiallyP}(f) \) and \( \text{InitiallyN}(f) \): means that \( f \) holds or does not hold, respectively, from \( t=0 \).

**B. Operations over Obligations**

Next, we will proceed with the formalization of the operations that can be applied over obligations and/or conditional obligations, indistinctively.

1. **Creating obligations**: The creation of an obligation to reach a state of affairs \( f \), by performing an action \( act() \), is shown next:

   \[
   \begin{align*}
   \text{CreateO}(act(a), O(a,f), t) : \\
   \{ \text{Happens}(act(a), t) \land \text{Initiates}(act(a), O(a,f), t) \}. 
   \end{align*}
   \]

   Where the predicate \( act(a) \) denotes that the action \( act() \) is performed by agent \( a \); the fluent \( O(a,f) \) represents that agent \( a \) is obliged to reach the state of affairs \( f \), and \( t \) represents the instant when the action is performed.

   An obligation can also be induced to another agent. This is defined as follows:

   \[
   \begin{align*}
   \text{CreateO}(act(a), CO(b, f, g), t) : \\
   \{ \text{Happens}(act(a), t) \land \text{Initiates}(act(a), CO(b,f,g), t) \land \\
   \text{HoldsAt}(g, t) \} \text{ where } g = \text{AllowedBy}(b, O(b,f)).
   \end{align*}
   \]

   If the predicate \( \text{AllowedBy}(b, O(b,f)) \) is evaluated true, implies that agent \( b \) accepts the obligation \( O(b,f) \) induced by agent \( a \).

2. **Releasing obligations**: The releasing of an obligation takes place when the state of affairs \( f \) has been reached by agent \( a \). This is stated as follows:

   \[
   \begin{align*}
   \text{ReleaseO}(act(a), O(a,f), t) : \\
   \{ \text{Happens}(act(a), t) \land \text{Initiates}(act(a), f, t) \land \\
   \text{Terminates}(act(a), O(a,f), t) \}. 
   \end{align*}
   \]

   In order to ensure an appropriate management of the creation and releasing of obligations, the following rules are required:

   \[
   \begin{align*}
   \text{CreateO}(act(a), O(a,f), t) & \leftarrow \text{HoldsAt}(O(a,f), t) \land \text{Happens}(act(a), t). \\
   \text{ReleaseO}(act(a), O(a,f), t) & \leftarrow \text{HoldsAt}(O(a,f), t) \land \text{Happens}(act(a), t).
   \end{align*}
   \]

   The first rule indicates that an obligation can be created only if it is not already held; the second rule states that an obligation can be released only if the obligation is held. More details about the obligation approach can be found in [10].

**III. WEB SERVICES EXPRESSED WITH OBLIGATIONS**

A web service \( W \) is a set of 3-tuples \( (P_i, A_i, E_i) \), where \( P \) stands for preconditions; \( A \) stands for action; \( E \) stands for effects, and \( i \in [1, |W|] \).

There are two kinds of preconditions:

- when a state of affairs has to be already accomplished, denoted by \( \text{HoldsAt}(f, t) \), and
- when it is enough that someone be committed to reach \( f \), denoted by \( \text{HoldsAt}(O(a,f), t) \).

The execution of an action, which alters the state of the interaction, is represented as a message among the interested parties. A message can be internal or external. Internal, when the aim of the message is to change the state of the agent; and external, when the action execution also affects the state of the interaction of another agent.

The effects of actions are represented by the releasing and creation of obligations.

An atomic web service consists of only one action, and a complex web service contains more than one action [13]. The order of execution of the actions will depend on the preconditions of each action, moreover, more than one action could be enabled at some point of the interaction, then, the agent in charge of the web service will select the best action to execute according to its convenience.

**A. Agent Infrastructure for supporting Obligation-based Web Services**

An agent that is in charge of an obligation-based web service requires additional elements in order to be capable of interacting with other agents. Our agents have three components:
TABLE I. SEARLE’S ILOCUTIONARY ACTS TAXONOMY

<table>
<thead>
<tr>
<th>Class</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertives</td>
<td>“Convince the hearer to the truth of the expressed proposition”.</td>
</tr>
<tr>
<td>Directives</td>
<td>“Attempts by the speakers to get the hearer to do something”.</td>
</tr>
<tr>
<td>Commissives</td>
<td>“Commit the speaker to some future course of action”.</td>
</tr>
<tr>
<td>Declarations</td>
<td>“Bring about some alteration in the status or condition of the referred to object”.</td>
</tr>
<tr>
<td>Expressives</td>
<td>“Express the psychological state specified in the sincerity condition about a state of affairs”.</td>
</tr>
</tbody>
</table>

a) Interaction context (IC): It contains the fluenets that agents can acquire as obligations, as well as, those ones that can be induced to other agents. A fluent can have four properties:

- **Self-induced** (\(\circ\)). It denotes that the fluent can be self-induced as obligation by agent's own decision.
- **Out-in induction** (\(\downarrow\)). It denotes that the fluent can be externally induced as obligation by other agent.
- **In-out induction** (\(\uparrow\)). It denotes that the agent could have the right to induce the fluent as an obligation to other agents.
- **Final** (\(\ast\)). It denotes that the fluent assigned as obligation, can be released by the agent, without intermediaries.

This also serves as a description of the functionalities of the web service, and it is considered to be associated to an OWL-S ontology [13], which is pre-agreed among agents.

b) Interaction state (IS): Its main function is to keep a record of the interaction, grouped by conversations. This is necessary, given that complex web services require the establishment of a conversation among the participant agents. Formally speaking, the interaction context of a conversation \(C\), for a given time \(t\), is defined as follows:

\[
\text{Interaction state}_{C} = \{\text{HoldsAt}(O(a,f),t) \cup \text{HoldsAt}(CO(b,g,h),t) \cup \text{HoldsAt}(i,t) \forall f, \forall g, \forall h, \forall i \in \text{Interaction context}\}.
\] (6)

The interaction state is updated by means of the emission and reception of messages.

c) Initial obligations (IO): An agent may commit to achieve a set of obligations by the mere fact of containing a particular web service. This is denoted as follows:

\[
\text{IO} = \{f_{1}, f_{2}, ..., f_{N}\}.
\] (7)

With the initial obligations established, we can define the initial interaction state of an agent, as follows:

\[
\text{InitiallyP}(O(a,f)) \forall f \in \text{IO} \cup \text{InitiallyN}(O(a,g)) \forall g \in (\text{IC} \setminus \text{IO}) \cup \text{InitiallyN}(CO(a,g,h)) \forall g \in (\text{IC} \setminus \text{IO}) \cup \text{InitiallyN}(h) \forall h \in \text{IC}.
\] (8)

We consider that the agent infrastructure is public, and that it can be accessed by any agent, but only the agent owner can modify it. This supports the agent interaction in open systems, such as the semantic web environment.

IV. THE OBIGATION-BASED COMMUNICATION LANGUAGE

As we mentioned before, our agent communication language is based on speech act theory. In order to define our primitive illocutionary acts, we follow the illocutionary act taxonomy presented in [16], which is summarized in table I.

We do not take into account expressive acts, since they do not apply to our approach. We make use of some FIPA-ACL primitives acts [7], but we replaced their BDI definition with a more appropriate definition based on obligations, which does not depend on agent beliefs and is suitable for open environments, where heterogeneous and untrustworthy agents are present. Next we present our illocutionary acts:

a) **Assertives**: The assertive illocutionary act attempts to convince the receiver of the message that some state of affairs \(f\), was achieved. This is stated as follows:

\[
\text{Inform}(a, b, f, \text{data}) \equiv \text{ReleaseO}(O(a,f),t).
\] (9)

Where \(a\) and \(b\) are agents that represent the sender and receiver of the message, respectively; \(f\) is a fluent that belongs to the interaction context; and the parameter \(\text{data}\) is additional information obtained as a result of achieving the state of affairs \(f\). On the obligation side, an Inform message has the effect of releasing an obligation to achieve a state of affairs, previously assigned to the sender.

When a message is sent, it alters the interaction state of both the receiver and the sender.

d) **Directives**: Directives acts attempt to induce obligations on other agents, by making use of conditional obligations, as we previously described in section II, therefore the directive illocutionary act is defined as follows:

\[
\text{Request}(a,b,f) \equiv \text{CreateO}(CO(b,f,g),t)\text{.}
\] (10)

A Request message has the effect of creating a conditional obligation to agent \(b\) to achieve \(f\) requested by agent \(a\). The fluent \(g\) represents agent \(b\)'s decision of accepting or not the obligation to achieve \(f\). Therefore, when a conditional obligation is created, it is not activated until the receiver accepts it.

c) **Commissives**: These illocutionary acts can be interpreted as a promise of an agent to achieve a certain state of affairs. We do not define a primitive act to achieve this, instead, we simulate it, by making use of a small protocol, that is to say, after a Request message its receiver could send an Agree message to promise some future course of action. This illocutionary act is defined next:
An Agree message has the effect of creating an obligation to achieve f requested by agent a, and at the same time, it releases the conditional obligation by means of which the fluent f was induced as obligation. In this way, agent b accepts the induction of the obligation by an external agent. However, if agent b does not accept the induction of the obligation, it sends a message of rejection, which is defined as follows:

\[ \text{Reject}(b, a, f) \equiv \text{Release}(O(b, f, g), t) \]  
where \( g = \text{AllowedBy}(b, O(b, f)) = \text{false} \).

Both Agree and Reject messages can only be sent after the reception of a Request message.

d) Declarations: The objective of declarations is to produce an alteration in the interaction state of the agents, in regard to the obligation referred to in the message. To avoid that an agent produces undesirable changes in other agents’ obligations, we limit declarations only to changes in the sender of the declaration. Next, the formal definition:

\[ \text{Confirm}(a, b, O(a, f)) \equiv \text{Create}(O(a, f), t) \]  
\[ \text{Disconfirm}(a, b, O(a, f)) \equiv \text{Release}(O(a, f), t) \]  
where \( a \) is the sender of the declaration.

A Disconfirm message has the effect of releasing the obligation \( O(a, f) \).

V. CONVERSATION-BASED SERVICE COMPOSITION

In order to make a service composition, the user makes use of a Composer Agent (henceforth CA), which is in charge of searching and composing services, using obligation-based conversations. Fig. 1 shows the architecture proposed.

The CA serves as a common interface among agents that could provide the web services, which satisfy the user requirements.

A. Composition Algorithm

Next, we present the algorithm followed to create the composer agent that will be capable of composing services:

1) The user assigns k goal obligations \( \{GO_1, GO_2, ..., GO_k\} \).

\[ \text{IO} = \{GO_1, GO_2, ..., GO_k\} \]  
(15)

3) The CA creates its interaction context \( \text{IC} = \{GO_i\} \) in-out induction property is assigned to all obligations, with the aim of inducing them to other agents, and the self-induced property is assigned, because the obligations are adopted by agent's own decision.

4) The CA does a search among available services, using the new goal obligations as search parameters, obtaining as a result a set of services \( S \), which can release at least one of the goal obligations.

5) The CA makes an appropriate union of interaction contexts, as follows:

\[ \text{IC}_{CA} = \text{IC}_{CA} \cup \text{NIC}_{i} \]  
where \( \text{NIC}_{i} \subset S.IC \) if \( f \in \text{NIC}_{i} \) and \( f \) has the in-out induction property \( \uparrow \), \( i \in [1, |S|] \).

The selection of fluents in such a way, means that the CA accepts the obligations induced by the corresponding agent. These fluents are added to its interaction context, but with the appropriate properties: out-in induction \( (\downarrow) \) to accept them, and in-out induction \( (\uparrow) \) to induce them to other web services, which may release them.

6) If all new added obligations can be released within the context of the participant agents, continue to step 7, otherwise, provide all the obligations that cannot be released as the new goal obligations and go back to step 4.

7) The CA plans a conversation with the participant agents, where the initial state is defined as follows \( \text{HoldsAt}(O(a, f), t) \) if \( f \in \text{IO} \) and the goal state will be given by any Interaction State that contains \( \text{HoldsAt}(f, t) \) if \( f \in \text{IO} \).

The planning process to obtain a conversation that achieves the goal state, can be solved by simple ordered planners, such as the presented in [14]. The planning process is out of the scope of this paper.
B. Composer Agent’s Web Service

We have already defined the CA’s infrastructure, now we proceed to define the web service contained in the CA. It has three different sorts of actions:

a) Since the CA does not release any obligation by itself, it requires inducing all the obligations of the interaction context to any agent b that could release them, then, whenever an obligation to achieve f holds, the CA sends a Request message. This is stated as follows:

\[
P_i : \text{HoldsAt}(O(CA, f), t) \\
A_i : \text{Request}(CA, b, f) \\
E_i : \{\emptyset\} \quad \forall f \in IC.
\]

(17)

b) Given that the CA serves as interface, it should agree or reject any Request messages received. The set of actions of this sort is defined next:

\[
P_k : \text{HoldsAt}(CO(CA, f, AllowedBy(CA, O(CA,f))), t) \\
A_k : \text{Agree}(CA, b, f) \\
E_k : \{\emptyset\} \quad \forall f \in (IC \setminus IO).
\]

(18)

If the CA agrees to create the obligation to achieve f, it has to make use of actions of the first group, in order to release the obligation accepted.

c) The third group of actions deals with updating the interaction contexts of the participant agents, thus, whenever a fluent f holds, the CA informs to the corresponding agents

\[
P_m : \neg\text{HoldsAt}(CO(CA,f,AllowedBy(CA, O(CA,f))), t) \\
A_m : \text{Reject}(CA, b, f) \\
E_m : \{\emptyset\} \quad \forall f \in (IC \setminus IO).
\]

(19)

The three groups of actions have no effects other than those derived in the definition of the illocutionary acts.

The previous sets of actions and the CA’s infrastructure defined, give as a result, a generic composer agent, which receives as input, goal obligations to create a conversation-based service composition.

VI. AN ILLUSTRATIVE EXAMPLE

The objective of this example is to show the feasibility of expressing web services and their interaction based on obligations, as well as to show the soundness provided by them.

We use the travel reservation use case presented in [11]. Here, we have four different kinds of web services:

- **Airline web service (A)** offers and makes flight reservations.
- **Bank web service (B)**, which is in charge of making payments on behalf of the user, previous an authentication process.

Agent T is in charge of the service composition process, with the objective of providing a solution to the user. Therefore, agent T will be considered to be the composer agent.

Next, we present the definition of agents T, H, A, and B. We provide only the messages that agents can send. Preconditions and effects are omitted by explanation purposes; nevertheless, these can be easily obtained, following the definition of the composer agent’s web service, presented in section V.

The automatic generation of the composer agent can be validated by simple inspection and following the algorithm presented in section V.

The users express their requirements by means of obligations, we assume that the user has access to the OWL-S ontology, and for this particular use case, the user selects the BookRoom and BookFlight obligations. The selection of these obligations, determines the structure of the composer agent. Next, the web service definitions:

**Travel agent web service (T)**

\[IC_T = \{\text{CheckAvailability}(\text{params}), \text{BookRoom}(\text{params}), \text{BookFlight}(\text{params})\}\]

**Bank web service (B)**

\[IO_T = \{\text{MakePayment}(\text{Params})\}\]

**Actions**

\[
A_1: \text{Request}(T, H, \text{CheckAvailability}(\text{params})) \\
A_2: \text{Request}(T, H, \text{BookRoom}(\text{params})) \\
A_3: \text{Request}(T, A, \text{SearchFlights}(\text{params})) \\
A_4: \text{Request}(T, A, \text{BookFlight}(\text{params})) \\
A_5: \text{Request}(T, B, \text{MakePayment}(\text{Params})) \\
A_6: \text{Request}(T, B, \text{BookFlight}(\text{params})) \\
A_7: \text{Agree}(T, H, \text{CheckAvailability}(\text{params})) \\
A_8: \text{Reject}(T, H, \text{CheckAvailability}(\text{params})) \\
A_9: \text{Agree}(T, H, \text{SearchFlights}(\text{params})) \\
A_{10}: \text{Reject}(T, H, \text{SearchFlights}(\text{params})) \\
A_{11}: \text{Agree}(T, H, \text{MakePayment}(\text{Params})) \\
A_{12}: \text{Reject}(T, H, \text{MakePayment}(\text{Params})) \\
A_{13}: \text{Agree}(T, A, \text{MakePayment}(\text{Params})) \\
A_{14}: \text{Reject}(T, A, \text{MakePayment}(\text{Params})) \\
A_{15}: \text{Inform}(T, H, \text{CheckAvailability}(\text{Params}, \text{data})) \\
A_{16}: \text{Inform}(T, A, \text{SearchFlights}(\text{Params}, \text{data})) \\
A_{17}: \text{Inform}(T, H, \text{MakePayment}(\text{Params}, \text{data})) \\
A_{18}: \text{Inform}(T, A, \text{MakePayment}(\text{Params}, \text{data}))
\]

**Hotel web service (H)**

\[IC_H = \{\text{CheckAvailability}(\text{params})\} \downarrow \cdot \text{BookRoom}(\text{params})\} \downarrow \cdot \text{MakePayment}(\text{Params})\} \uparrow\]

\[IO_H = \{\emptyset\}\]
Actions_{H} = 
A_{1}:: Agree(H, T, CheckAvailability(Params)) 
A_{2}:: Agree(H, T, BookRoom(Params)) 
A_{3}:: Reject(H, T, CheckAvailability(Params)) 
A_{4}:: Reject(H, T, BookRoom(Params)) 
A_{5}:: Inform(H, T, CheckAvailability(Params), data) 
A_{6}:: Inform(H, T, BookRoom(Params), data) 
A_{7}:: Request(H, T, MakePayment(Params))

Bank web service (B)

IC_{B} = \{MakePayment(Params)\} \uparrow, \{BookFlight(params)\} \downarrow, \{MakePayment(Params)\} \uparrow | IO_{B} = \{\emptyset\}

Actions_{B} =
A_{1}:: Agree(B, T, MakePayment(Params)) 
A_{2}:: Reject(B, T, MakePayment(Params)) 
A_{3}:: Inform(B, T, MakePayment(Params), data)

A possible conversation among these agents is the following:

<table>
<thead>
<tr>
<th>Action ID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_{A3}</td>
<td>Request(T, A, SearchFlights(Params))</td>
</tr>
<tr>
<td>T_{A1}</td>
<td>Request(T, H, CheckAvailability(Params))</td>
</tr>
<tr>
<td>A_{1}</td>
<td>Agree(A, T, SearchFlights(Params))</td>
</tr>
<tr>
<td>A_{1}</td>
<td>Agree(H, T, CheckAvailability(Params))</td>
</tr>
<tr>
<td>A_{3}</td>
<td>Agree(H, T, CheckAvailability(Params))</td>
</tr>
<tr>
<td>A_{5}</td>
<td>Agree(H, T, CheckAvailability(Params), data)</td>
</tr>
<tr>
<td>H_{5}</td>
<td>Agree(H, T, CheckAvailability(Params), data)</td>
</tr>
<tr>
<td>T_{A2}</td>
<td>Request(T, H, BookRoom(Params))</td>
</tr>
<tr>
<td>H_{6}</td>
<td>Agree(H, T, BookRoom(Params), data)</td>
</tr>
<tr>
<td>T_{A4}</td>
<td>Request(T, A, BookFlight(Params))</td>
</tr>
<tr>
<td>A_{2}</td>
<td>Agree(A, T, BookFlight(Params))</td>
</tr>
<tr>
<td>A_{6}</td>
<td>Agree(A, T, BookFlight(Params), data)</td>
</tr>
<tr>
<td>A_{7}</td>
<td>Agree(A, T, BookFlight(Params), data)</td>
</tr>
<tr>
<td>T_{A3}</td>
<td>Request(T, A, MakePayment(Params))</td>
</tr>
<tr>
<td>T_{A5}</td>
<td>Request(T, B, MakePayment(Params))</td>
</tr>
<tr>
<td>B_{A1}</td>
<td>Agree(B, T, MakePayment(Params))</td>
</tr>
<tr>
<td>B_{A1}</td>
<td>Agree(B, T, MakePayment(Params))</td>
</tr>
<tr>
<td>T_{A17}</td>
<td>Inform(T, H, MakePayment(Params), data)</td>
</tr>
<tr>
<td>T_{A5}</td>
<td>Request(T, B, MakePayment(Params))</td>
</tr>
<tr>
<td>B_{A1}</td>
<td>Agree(B, T, MakePayment(Params))</td>
</tr>
</tbody>
</table>

This conversation complies with \( \text{HoldsAt}(f, t) \mid \forall f \in \text{IO}_{T} \), that is to say, all the goal obligations of the composer agent \( T \) hold at the end of the conversation, achieving a successful web service composition. Nevertheless, when the conversation is enacted at runtime, some agents could reject the induction of certain obligations, due to different reasons. Therefore, the composition process presented here, obtains a possible solution, if there is one, but it does not guarantee its successful execution, since it depends on agents’ autonomy.

The example illustrates the ease in defining, either atomic or complex web services, by making use of obligations. In addition, the interaction state of a conversation among a set of agents, is maintained individually, and can be determined at any time \( t \). Moreover, it shows the advantages of the incremental approach used to define web services, which allows adding or removing actions in an easy way. Finally, this example also showed the ease in instantiating the generic composer agent that serves as the interface to combine semantic web services.

VII. RELATED WORK

There are some approaches, which are close to ours. With respect to the agent communication languages, we have the work presented in [8], where a commitment-based ACL is defined. They make use of commitment objects to provide semantics to speech acts, however there is not an explicit support to keep track of the commitments created between agents, in order to determine the interaction state for a given point of time. Additionally, the main assertive speech act (inform) and the main commissive act (promise) have the same explicit definition, leaving their differences to the interpretation, obtaining ambiguous definitions. Another difference with our work, is the definition of a promise primitive act for commissives, meanwhile, we manage commissives as the response of the reception of a directive. We consider this better reflects the interaction among agents.

Another work that proposes commitment semantics for ACLs is presented in [18]; here a commitment requires the participation of three agents: a debtor, a creditor, and an authority, which validates the commitment. This meaning of commitment has similarities with the concept of obligation presented in this paper. We consider obligations as self-commitments, i.e., an agent commits itself to achieve a state of affairs, but without requiring a creditor. In an obligation, the debtor and the creditor are represented by the same agent. This extends the scope of validity of obligations to all the contexts of interaction of the agent, and not only to the context of its creditor, as in commitments. Nevertheless, obligations require external mechanisms to guarantee the fulfillment of agents’ obligations.

With regard to the agent conversation-based web service composition, the closest research to ours is the work presented in [2], where semantic web services are defined as a set of actions, with preconditions and effects; in addition, they present a high level ACL based on modal logic and
mental states of agents, expressed by means of beliefs, being this last aspect, the one that limits the deployment of their approach in open environments, where heterogeneous agents are present.

VIII. CONCLUSIONS AND FUTURE WORK

In this work, we delineated semantic web services by means of obligation fluents. We also defined an agent communication language, composed of a set of illocutionary acts, with semantics expressed in terms of obligations.

We argue that our obligation-based conversation approach is suitable for supporting semantic web service composition. Our main contribution to this issue is the design of a conversation-based web service composition process, which automatically generates a generic composer agent, which is directed by a set of goal obligations that represents the user requirements. This composer agent takes users’ requirements as goal obligations that direct the search and composition of web services, once the participant services are selected, the composer agent manages the interaction among agents.

Our future work will be addressed to define more illocutionary acts, using obligation semantics; as well as developing mechanisms capable of recovering conversations from failed states (e.g. rejection of an obligation), by means of introducing an argumentation framework.

ACKNOWLEDGMENT

This research is partially supported by CoECyT-Jal project no. 2008-05-97094, whilst author J. Octavio Gutiérrez-García is supported by CONACYT grant no. 191493.

REFERENCES