A Framework of Open Interactions based on Web Services and Semantic Web Technologies

Nicoletta Fornara[1⋆], Marco Colombetti[2⋆⋆], Daniel Okouya[1⋆⋆⋆]

[1] Università della Svizzera italiana,
via G. Buffi 13, 6900 Lugano, Switzerland
{nicoletta.fornara, daniel.okouya}@lu.unisi.ch,
[2] Politecnico di Milano, piazza Leonardo Da Vinci 32, Milano, Italy
marco.colombetti@polimi.it

Abstract. The design and implementation of open interaction systems (OIS) is widely recognized to be a crucial issue in the development of innovative applications on the Internet. In our past works we have proposed to specify the high-level component of an OIS by means of a set of artificial institutions. In this paper we describe the lower layers of a framework of open interaction, to be used as a reference for the development of an OIS: the connectivity layer, in charge of enabling an agent to send messages to other agents, and the message layer, devoted to the syntactic and semantic processing of the messages. We specify a set of conventions concerning these layers that are essential in open interactions: we propose conventions on the message structure, on the application independent part of the content language, and on the semantics of messages (limiting our attention, in this paper, to commissive messages). Such conventions would guarantee agent interoperability if adopted as interaction standards. Finally we present a software architecture based on Web Service and Semantic Web Technology, suitable for the actual implementation of these layers, and a demonstrative implementation.

1 Introduction

The design and implementation of *open interaction systems* (OIS) is widely recognized to be a crucial issue in the development of innovative applications on the Internet, like e-commerce applications, e-marketplaces, and applications for the management of virtual enterprises. An interaction systems is *open* when

⋆ Supported by the Swiss State Secretariat for Education and Research SER within the COST action IC0801 "Agreement Technologies", project n. C08.0114, title "Open Interaction Frameworks: A Model Based On Artificial Institutions".

⋆⋆ This research has been carried out as part of the activities of the author at Università della Svizzera italiana, Switzerland.

⋆⋆⋆ Supported by the Swiss National Science Foundation project n. 200020_134790, title "Open Interaction Systems: An approach based on Artificial Institutions and Service Oriented Architecture with an application to Electronic Marketplaces".

different programs (regarded as autonomous agents) may freely enter or leave the system, and try to achieve their individual goals by interacting with other agents according to shared rules.

In our past works [7, 6, 4] we have proposed to specify the high-level component of an OIS by means of a set of *artificial institutions*(specified using the OCeAN meta-model), that is, as a system in which agents play certain roles, are empowered to perform certain actions, and are subject to a set of norms regulating the interactions. In the work presented in this paper we start our analysis from the opposite side, that is, from the lower layers that are necessary to achieve *interoperability* among two o more agents that are developed by different designers and are running on different platforms. This choice is justified by the aim of testing our proposal with a demonstrative implementation that will be presented in Section 5. As we will discuss in Section 4, an important component of our proposal is the realization of an *OIS Manager*, which will be in charge of managing complex interactions, like for example collective activities, among different autonomous agents running on different platforms.

We start from a bottom-up analysis of the goals, assumptions, and constrains for the realization of open interactions. This analysis brought us to identify four *layers* for the representation of open interaction, where the concepts of the lower layers contribute to the definition of the concepts belonging to the higher layers. The layers we identified are: the *connectivity layer*, in charge of enabling an agent to send messages to other agents; the *message layer*, which presupposes connectivity and specifies a suitable Agent Communication Language (ACL); the *conversation layer*, which presupposes the ability to exchange messages in a given ACL, and specifies rules for carrying out conversations; and the *activities layer*, where complex activities going beyond pure communication are carried out (e.g., e-business interactions). As we want to test our proposal with a demonstrative implementation, and due to space limitations, in this paper limit our analysis to the two lower layers: the connectivity and the message layers. We plan to investigate the upper layers in our future works.

In particular in Section 2 we propose to analyze the connectivity and message layers starting from the study of the assumptions and requirements that we have to take into account for the realization of open interaction. This with the goal of specifying a *general framework* of open interaction that should be sufficiently general, that is, independent of specific application domains and enabling technologies. We continue our work in Section 3 by defining a minimal set of *conventions* concerning the various components of the general framework, that would guarantee agent interoperability if adopted as interaction standards. In Section 4 we presents a software architecture, suitable for the actual implementation of an open interaction system exploiting software technologies that can be readily adopted or, even better, that have already been adopted by the industry, like Semantic Web and Web Service technologies. Finally, in Section 5 we describe a demonstrative implementation of the two bottom communication layers of the architecture.

2 A Framework of Open Interaction

In this section we describe the two lower layers of a framework to be used as a reference for the development of an OIS. The framework proposed is general, in the sense that it makes no reference to specific application domains or underlying conventions and technologies. Our proposal is based on a number of assumptions:

1. An agent engages in *interactions* with other agents in the attempt to achieve its individual goals. These agents typically run on different and possibly heterogenous *platforms*;
2. Interactions take place in an application-dependent *interaction context*. Intuitively, the interaction context includes all resources that are necessary to carry out the interactions within a well-defined application domain.
3. All events belonging to the interaction, and all resources belonging to the interaction context, satisfy a set of application-independent *conventions* (e.g., the syntax and semantics of messages, and so on).
4. In interactions, individual agents try to achieve their individual goals by carrying out certain *collective activities*; for example, such activities may consist of complete commercial transactions. Such activities are carried out through processes of communication: agents engage in conversations, realized by exchanging messages with shared syntax, terminology, and semantics. In turn, message exchange presupposes suitable connectivity.

These assumptions introduce a number of concepts that we now organize bottom-up in a layered way, limiting our analysis to the two lower layers.

Layer 1: Connectivity. Connectivity is defined as the ability of an agent to send messages to other agents, and to receive messages from them: providing connectivity implies that, in normal conditions, a message sent by Agent1 (the *sender*) to Agent2 (the receiver) will actually be received by Agent2 within acceptable time. Connectivity presupposes the adoption of suitable *transport protocols*. Moreover, we assume that each agent has a well-defined identity, which is kept constant across different interaction. This justifies the assumption that every agent has a unique *agent identifier*. The fact that it is unique implies the existence of a service for the registration of unique names, similar the the service for domain name registration in Internet. We do not assume that such an identifier is initially known to all possible partners: this implies the need of *agent directories*. Both transport protocols and the structure and use of agent directories and agent registration services are specified at the level of the underlying software architecture (see Section 4).

Layer 2: Messages. We say that a message is *exchanged* when it is sent by a sender and actually received by the intended receiver. The ability to exchange messages presupposes both connectivity (Layer 1) and the adoption of a suitable *Agent Communication Language* (ACL), which specifies: (i) the logical structure of messages (i.e., the *abstract syntax* of messages, independent of both the application domain and the underlying software architecture); (ii) the contribution of the message structure to the meaning of the message. As usual, the *semantics* of a specific message will be defined compositionally, that is, by suitably

combining the semantics of application-dependent terminology with the semantics of the application-independent elements together with the structure of the message. This presupposes that: (i) the syntax and semantics of the application-independent components of the ACL, together with the syntax (but not the semantics) of the application-dependent terminology, are defined as suitable conventions; (ii) the semantics of application-dependent terminology is represented in *domain ontologies*, which are part of the interaction context.

3 Conventions for the Message Layer

An OIS is a system where agents can interact to achieve their individual goals by performing complex activities. Contrary to more traditional distributed systems, the distinctive feature of an OIS is that it allows external agents to enter the system, participate in the activities, and then leave the system at will. This brings to the foreground the problem of interoperability, because for the interaction to be successful all agents have to comply to a set of *shared conventions*, which will have to be taken as *standards* by the designers of the agents. In this section we introduce a set of possible conventions, named OCeAN-ACL, covering the various parts of the message layer: the structure of the messages (i.e. their *abstract syntax*), the application independent part of the *content language*, and the *semantics*.

Regarding the structure of messages and the content language, when possible we will try to roughly follow FIPA-ACL specifications[3]. Regarding semantics, as discussed in [10, 5], we depart more significantly from the semantics of FIPA-ACL, which presents the problem of not taking into account the normative consequences of message exchanges. More specifically, we adopt commitment-based semantics for communicative acts and institutional semantics for declarations. In the past we have expressed the semantics of communicative acts using the Event Calculus [6]. In this paper, given that we want to exploit industry-level technology, we adopt OWL (more precisely, OWL 2 DL[4]) to express the semantics of communicative acts. For the moment we only propose a semantics of commissive communicative acts, using an extension of the OWL ontology of obligations presented in [4]. We plan to further extend this ontology in the future, to express the semantics of directive and assertive communicative acts, and the semantics of declarations.

3.1 Message Structure

In this section we propose the message structure of our ACL. Like in FIPA-ACL, a message contains an acl-envelope, used to correctly route the message to the final destination and comprised of a number of attributes, including to, from, length, encoding, and others. The message also contains an acl-payload, characterized by the following components:

[3] http://www.fipa.org/repository/aclspecs.html
[4] http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

- performative: a symbol denoting the type of the communicative act, that in our ACL may be: promise, inform, request, agree, refuse, cancel, query-ref, query-if, and declare; further types may be added and specified according to need;
- sender: the identifier of the agent that sends the message;
- receiver: the identifier of the recipients of the message;
- content: a complex expression that can represent: a state of affair if the performative is assertive (like inform), an action with a deadline if the performative is commissive (like promise) or directive (like request), and an institutional action if the performative is declare;
- reply-by: the time within which it is possible to answer to a request;
- msg-id: the unique identifier of the message, generated by the sender of the message by using its name as namespace followed by a progressive number, for example John:001.

The following three parameters are used to describe the content of the message: (i) language denotes the formal language of the content expression; (ii) encoding denotes the specific encoding of the content language; (iii) ontology denotes the ontology used in the content of the message.

FIPA-ACL provides for other components, like reply-to, protocol, conversation-id, reply-with, in-reply-to; as these are used to control the conversation, they belong to the conversation layer and will not be treated in this paper.

3.2 Content language

A *content language* consists of: (i) a set of fundamental *application independent* terms, like for example those used to describe *actions* and *propositions*; and (ii) a set of *application dependent* terms used in the content of messages related to specific domains, for example to describe events of payment, delivery, bidding and so on. Both application-independent and application-dependent terms need to be connected by a formal language.

We base our formal language on RDF[5] and RDFS[6] because they have been recommended by W3C since 2004, are widely adopted for representing knowledge that can be re-used in specific applications, and are expressive enough to represent resources, classes, and properties that are the main concepts that need to be communicated in the content of a message. Given that RDF has a formal semantics it is possible to perform reasoning on the content of messages by simply sharing application dependent ontologies.

In this section we will describe the application independent concepts that can be used in the specification of the content of a communicative acts. In doing so we started from FIPA-RDF Content Language fipa-rdf0, but extended it with other concepts and properties that we think are general enough to be part of the application-independent component of our language.

[5] http://www.w3.org/TR/rdf-syntax/
[6] http://www.w3.org/TR/rdf-schema/

Similarly to what is proposed in [2] the content of a message can be a proposition, or the description of an action, or the description of an action together with a condition event, or an expression containing variables for expressing for example query-ref communicative acts. In order to be able to represent such a different contents in our ontology we introduce the ocean:ContentElement class that has different subclasses used for modeling the different types of allowed contents on the basis of the types of the communicative act. One subclass is the class ocean:ConditionalActionDesc used for representing contents with an action and a condition event, which are necessary for example for *request* or *promise* communicative acts. The content and the condition part are inserted in the message by introducing the property ocean:hasContentPart, whose range is the ocean:ActionDesc class, and the property ocean:hasConditionPart, whose range is the ocean:EventDesc class (those classes will be better described in the following paragraphs). Another subclass of the ocean:ContentElement class is the rdf:Statement class used for assertive communicative acts. Other subclasses have to be defined for being able to *query* for some information or *declaring* the performance of an institutional action. In our content language the content of a message is connected to the message by means of the ocean:hasContent property. In this paper we will describe the classes for representing the notions of *proposition*, *action*, and *conditional event*.

Propositions. The content of an inform message is the description of a state of affairs, that is a *proposition*. RDF triples (composed by a subject, a predicate, and an object) are a good candidate for representing states of affairs. However, an agent may want to communicate information about propositions, for example "John has said that Mary has paid € 5 to John"; to this purpose it is preferable to use the class rdf:Statement, introduced in RDF for representing the reification of a proposition. A resource belonging to the rdf:Statement class has the properties rdf:subject, rdf:predicate, and rdf:oject. For example the previous proposition can be expressed in RDF as[7]:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix app: <http://www.people.lu.unisi.ch/fornaran/schemas/app-rdf0#>.
app:sentence1 rdf:type rdf:statement;
   app:author app:John;           rdf:subject app:Mary;
   rdf:predicate app:Pay;         rdf:object app:John.
```

Actions. The content of a request or promise communicative act includes the description of an *action*, or more precisely the description of a class of actions that ought to happen in the future. The semantics behind the exchange of a promise (as we will see in the Section 3.3) is that the sender will be in charge of performing, within a given deadline, a particular action belonging to the class of actions described in the message.

Therefore, in the application-independent component of the content language, it is necessary to represent the description of actions. Similarly to the

[7] In this paper we use for our examples the Turtle syntax of RDF, http://www.w3.org/TeamSubmission/turtle/.

definition presented in fipa-rdf0, we introduce in our application-independent language ocean-rdf0 (defined by means of an RDF Schema) the ocean:ActionDesc class. The content of a message requiring an action description must belong to such a class or to one of its subclasses. Such subclasses have to be defined in an RDF Schema used to define the application-dependent part of the content language, that is, the application-specific concepts used in the messages. The ocean:ActionDesc class is the domain of the following properties that can be used to characterize the action that has to be performed:

- The property ocean:hasActorDesc, used to represent the agent responsible for the execution of the action; the range of this property is the application-independent class ocean:Agent.
- The property ocean:hasDeadlineDesc, representing the time instant within which the action has to be performed. It is empty if the deadline for the performance of the action is not known when the message is sent. In this case the deadline depends on the time when some other event happens; for example the promise to pay a given book within one week from its reception.
- The property ocean:hasDurationDesc can be used to specify the interval of time within which the action has to be performed, starting from the instant of time when the event described in the condition component of the message will happen.
- Other properties can be defined and communicated for specific class of actions. For example, in an application-dependent language app-rdf0 it is possible to define the class app:PayDesc as subclass of the class oecan:ActionDesc with the property app:hasRecipientDesc, whose range is the ocean:Agent class, and the property app:hasAmountDesc for expressing the amount of money that has to be paid.

Conditional Event. In some cases it may be necessary to communicate also a *condition event* that describes a class of events, such that, as we will see in the section about semantics, if one event belonging to that class or to its subclasses happens within a certain deadline, the obligation to perform the promised or requested action becomes active. To communicate the description of events in the content of messages, we introduce in ocean-rdf0 the application-independent class ocean:EventDesc. This is a generalization of the class ocean:ActionDesc; in fact an action is regarded as an event with an actor, therefore the class ocean:ActionDesc is a subclass of ocean:EventDesc. The class ocean:EventDesc has other subclasses, in particular the class oceanTimeEventDesc that can be useful to represent as condition event the elapsing of a given instant of time.

An example: the promise communicative act. To exemplify the proposed message layer we formalize a type commissive act, *promise*. We use as example the following promise: "agent John promises to agent Mary that John will pay 5 euro to Mary within 2 days from the reception of book1". The message with the content expressed in RDF that uses the RDF Schemas previously introduced is:

```
(promise
 :sender John           :receiver Mary
 :language ocean-rdf0   :ontology app-rdf0
 :msg-id John:001
 :content
  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
  @prefix app: <http://www.people.lu.unisi.ch/fornaran/schemas/app-rdf0#>.
  @prefix ocean: <http://www.people.lu.unisi.ch/fornaran/schemas/ocean-rdf0#>.
  @prefix ex1: <http://www.people.lu.unisi.ch/fornaran/example/promise#>.
  ex1:John:001 ocean:hasContent ex1:content01.
  ex1:content01 rdf:type ocean:ConditionalActionDesc;
      ocean:hasContentPart ex1:JohnAction1;
      ocean:hasConditionPart ex1:MaryAction1.
  ex1:JohnAction1 rdf:type app:PayDesc;
      ocean:hasActorDesc "John";         ocean:hasDurationDesc "2";
      app:hasRecipientDesc "Mary";       app:hasAmountDesc "5".
  ex1:MaryAction1 rdf:type app:DeliverDesc;
      ocean:hasActorDesc "Mary";         app:hasRecipientDesc "John";
      app:hasObjectDesc "book1".
```

Where the app:DeliverDesc class is a subclass of the ocean:ActionDesc class with the property app:hasRecipientDesc whose range is the ocean:Agent class, and the property app:hasObjectDesc for describing the object that has to be delivered.

3.3 Semantics

The semantics of a message is built compositionally from three types of elements: (i) content-independent elements (like performative, sender, receiver, reply-by, and so on); (ii) application-independent components of the content (like proposition or action); and (iii) application-dependent components. The semantics we adopt in this paper is the commitment-based semantics that we firstly presented in [5], enriched with the semantics of declarations expressed via institutional concepts as presented in [6].

Every communicative act is characterized by a set of *preconditions* that need to be evaluated for the successful performance of the act and a set of *effects* that change the state of the interaction if the act is successfully performed. We represent the state of the interaction and the events and action that happen in the system using an OWL 2 DL[8] ontology. OWL 2 DL can be regarded as a decidable fragment of First-Order Logic (FOL), in particular it is a practical realization of a Description Logic known as $\mathcal{SROIQ}(\mathcal{D})$. It allows one to define *classes*, *properties*, and *individuals*. An OWL ontology consists of: a set of class axioms that specify logical relationships between classes (the TBox); a set of property axioms to specify logical relationships between properties (the RBox); and a collection of assertions that describe individuals (the ABox).

[8] http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

The choice of using OWL 2 DL for expressing the state of the interaction and the semantics of messages may seem in contrast with the proposal to use RDF for representing the content of messages. We prefer to use RDF as content language, because it has been already proposed by FIPA as content language, it is more widely used by programmers, and it is much simpler than OWL. Though, RDF is too weak to allow for interesting reasoning, and for this reason we prefer to use OWL as a knowledge representation language. In principle, however, an agent may use any formal language to represent its own knowledge base.

We now describe the application-independent part of the ontology that we propose to use for representing the events/actions that happen in the interaction, for example the communicative acts, and the semantics of communicative acts. This part defines the classes Agent, Action, Event, Obligation where Action $\sqsubseteq$ Event and Obligation $\sqsubseteq$ Event, it includes some classes from the *OWL Time Ontology* (like TemporalEntity, Instant, Interval), and has a collection of data and object properties. For example the atTime property relates an eventuality, like for example an event or an obligation, to a temporal entity, like for example an instant of time. The properties evSameTime and evBefore relates an eventuality to another eventuality on the basis of the time when they happen. This ontology defines also a set of axioms and SWRL (Semantic Web Rule Language)[9] rules for reasoning on the state of the interaction. Moreover thanks to the use of an external Java program that is in charge of representing the elapsing of time and the functionalities necessary to perform close world reasoning on certain classes, it is possible to monitor the state of obligations and react to their fulfillment or violation. In particular an obligation may be fulfilled by any individual action that belongs to a class of possible actions. This framework and the ontology is presented in [4].

In this paper we extend this ontology, originally devised for the representation of events and actions and for the representation and monitoring of obligations, with the components required for representing the exchange of messages and their semantics. Fist of all we add the class Message for representing the exchanged messages as individuals named with the msg-id of the message. We introduce the following properties that are used to connect a message with its components: hasPerformative: Message $\rightarrow$ Performative, where Performative is the class used to represents all the possible illocutionary forces of messages; hasSender: Message $\rightarrow$ Agent; hasReceiver: Message $\rightarrow$ Agent. The axiom Message $\sqsubseteq$ =1 hasPerformative $\sqcap$ =1 hasSender $\sqcap$ =1 hasReceiver, is used to express that a message has exactly one sender, one receiver and one performative. Another property is hasContent: Message $\rightarrow$ ContentElement, where ContentElement is the class introduced for representing the different types of possible contents of a message and is defined coherently to the definitions introduced in ocean-rdf0 RDF schema presented in the previous section. The exchange of a message generates an event of type ExchMsg $\sqsubseteq$ Action that happens at a certain instant of time and is connected to the message by means of the property hasMessage: ExchMsg $\rightarrow$ Message.

[9] http://www.w3.org/Submission/SWRL/

Following the commitment-based semantics for ACL proposed in [6], the performance of communicative acts has the effect of creating or changing social commitments. A commitments to the performance of an action is a special case of social commitment, and coincides with the obligation to perform the action. If the exchange of a message has the effect of creating an obligation, the obligation is connected to the event of exchanging the message by means of the following property: hasSource: Obligation → ExchMsg.

Semantics of the promise communicative act. As we already said every communicative act is characterized by a set of *preconditions* that need to be evaluated for the successful performance of the act and a set of *effects* that change the state of the interaction if the act is successfully performed. For a *promise* to be a valid communicative act it is necessary that: (i) the content describes a class of actions, that is, a subclass of the class ActionDesc; (ii) an instance of the action has to be performed in the future (i.e., if a deadline is specified it has to be in the future of the message exchange); (iii) the sender of the message is identical to the actor of the promised action; and (iv) if the promise is conditional, the condition is specified as a class of events, that is, as a subclass of the class EventDesc. The *effects* of a promise are to create an obligation of the sender of the message, directed to the receiver, to perform a certain action within a given deadline if a certain condition hold. Going into details, to represent the promise message introduced as an example in the previous section, we need to insert in the ontology the following ABox assertions:

Message(John:001), Performative(promise), hasPerformative(John:001,promise),
Agent(John), hasSender(John:001,John),
Agent(Mary), hasReceiver(John:001,Mary),
ConditionalActionDesc(content01), hasContent(John:001,content01);
PayDesc(JohnAction1), hasContentPart(content01,JohnAction1),
hasActorDesc(JohnAction1,John), hasDurationDesc(JohnAction1,2),
hasRecipientDesc(JohnAction1,Mary),
Thing(5euro), hasAmountDesc(JohnAction1,5euro),
DeliverDesc(MaryAction1), hasConditionPart(content01,MaryAction1),
hasActorDesc(MaryAction1, Mary), hasReceipientDesc(MaryAction1, John),
Object(book1), hasObjectDesc(MaryAction1,book1).

The event of type ExchMsg that represents the exchange of the message is represented by the following assertions (where instant1 represents the instant of time when the message is received):

ExchMsg(eJohnMary1), hasMessage(eJohnMary1,John:001),
Instant(instant1), atTime(eJohnMary1,instant1),

The obligation generated by this message is represented as follows:

Obligation(obl-1), hasSource(obl-1,eJohnMary1), atTime(obl-1,instant1),
hasDebtor(obl-1,John), hasCreditor(obl-1,Mary), ProperInterval(interval1),
hasInterval(obl-1,interval1), hasDurationDescription(interval1,duration1),
days(duration1, 2), where integer 2 represents the duration of the interval communicated with the message.

The class of events that may activate the obligation is equivalent to the class defined as the intersection of the class representing the event in condition part of the message with other classes defined on the basis of the properties of such an event.

$\mathsf{StartEvent\text{-}1} \equiv \mathsf{Deliver} \sqcap \mathsf{hasActor} \ni \mathsf{Mary} \sqcap \mathsf{hasRecipient} \ni \mathsf{John} \sqcap \mathsf{hasObject} \ni \mathsf{book1}$

The content class of the obligation is defined as the intersection of the class representing the action of the content part of the message with other classes defined on the basis of the properties of such an action.

$\mathsf{Content\text{-}1} \equiv \mathsf{Pay} \sqcap \mathsf{hasActor} \ni \mathsf{John} \sqcap \mathsf{hasRecipient} \ni \mathsf{Mary} \sqcap \mathsf{hasAmount} \ni \mathsf{5euro}.$

The following SWRL rule is introduced to deduce the instant of time when the interval of obl-1 starts; this is the instant of time when an individual belonging to the StartEvent-1 class happens:

$$\mathsf{StartEvent\text{-}1}(?e) \wedge \mathsf{atTime}(?e,?inst) \wedge \mathsf{hasInterval}(\mathsf{obl\text{-}1},?int) \rightarrow \mathsf{hasBeginning}(?int,?inst)$$

Once the beginning instant and the duration of the interval are known, it is possible to use the following SWRL rule, which uses the swrlb:add built-in, to deduce the value of the end instant of the interval (we assume that the duration of the interval is expressed in days):

$$\mathsf{hasBeginning}(?int,?inst1) \wedge \mathsf{inDateTime}(?inst1,?dt1) \wedge \mathsf{dayOfYear}(?dt1,?day1) \wedge \mathsf{Instant}(?int2) \wedge \mathsf{inDateTime}(?inst2,?dt2) \wedge \mathsf{dayOfYear}(?dt2,?day2) \wedge \mathsf{hasDurationDescription}(?int,?d) \wedge \mathsf{days}(?d,?value) \wedge \mathsf{swrlb{:}add}(?day2,?day1,?value) \rightarrow \mathsf{hasEnd}(?int,?inst2)$$

The Deadline-1 class contains only the time event that happens at the instant of time when the interval of the obligation finishes (that may be known or unknown when the obligation is created), as stated in the following axiom:

$$\mathsf{Deadline\text{-}1} \equiv \exists\ \mathsf{atTime}.(\exists\ \mathsf{hasEnd}^{-}.(\mathsf{hasInterval}^{-} \ni \mathsf{obl\text{-}1}))$$

For those obligations whose deadline event is a fixed time event, it is important to check that the start event happens before the end event. By introducing the following axiom the ontology becomes contradictory if the deadline time event is before or equal to the start time event:

$$\mathsf{Deadline\text{-}1} \sqcap (\mathsf{evBefore.StartEvent\text{-}1} \sqcup \mathsf{evSameTime.StartEvent\text{-}1}) \sqsubseteq \bot$$

When a new obligation is created it is necessary to introduce in the TBox the following four axioms necessary to deduce the state of a given obligation, that is, to deduce if the obligation belongs to the Activated, Cancelled, Fulfilled, or Violated classes. In order to be able to perform some form of closed world reasoning on the Cancelled and Fulfilled classes, the Java program sets the extension of the class KCancelled $\sqsubseteq$ Cancelled equivalent to the class that contains all obligations that are known to be in the Cancelled class, and set the class KFulfilled $\sqsubseteq$ Fulfilled equivalent to the class that contains all obligations that are known to be in the Fulfilled class.

$$\{\mathsf{obl\text{-}1}\} \sqcap \neg\ \mathsf{KCancelled} \sqcap (\exists \mathsf{evBefore}.(\mathsf{StartEvent\text{-}1} \sqcap \exists \mathsf{atTime.Elapsed}) \sqcup \exists \mathsf{evSameTime}.(\mathsf{StartEvent\text{-}1} \sqcap \exists \mathsf{atTime.Elapsed})\ ) \sqsubseteq \mathsf{Activated}$$

$$\{\mathsf{obl\text{-}1}\} \sqcap \exists \mathsf{evBefore}.(\mathsf{EndEvent\text{-}1} \sqcap \exists \mathsf{atTime.Elapsed}) \sqsubseteq \mathsf{Cancelled}$$

$$\{\mathsf{obl\text{-}1}\} \sqcap \mathsf{Activated} \sqcap (\exists \mathsf{evBefore}.(\mathsf{Content\text{-}1} \sqcap \exists \mathsf{atTime.Elapsed}) \sqcup$$

∃evSameTime.(Content-1 ⊓ ∃atTime.Elapsed)) ⊓ ∃evBefore.(Content-1 ⊓ ∃evBefore.Deadline-1) ⊑ Fulfilled

{obl-1} ⊓ Activated ⊓ ¬KFulfilled ⊓ ∃evBefore.(Deadline-1 ⊓ ∃ atTime.Elapsed) ⊑ Violated

4 Software architecture

In this section we propose a software architecture for OISs. We base our proposal on the principles and standards of Service Oriented Architecture (SOA), because they address crucial low-level aspects of openness and interoperability[3], are sufficiently mature and relatively stable, and are already accepted and used by a large industrial community.

Layer 1: Connectivity. Connectivity is guaranteed by current SOA standards. In particular the transport protocol is HTTP, and the message structure protocol is SOAP[10](Simple Object Access Protocol). In the most popular implementation of the Service Oriented Architecture two distinct software applications play the role of *Service Requestor* and the role of *Service Provider*. This architecture can be adapted to our need of peer to peer (P2P) interactions by merging the two roles into one software application that plays simultaneously both roles.

Layer 2: Messages.

In order to be exchanged messages presented in Section 3, are structured according to SOAP standard. In particular both the envelope and the payload of OCeAN-ACL agent messages are serialized as an XML document and they constitute the body of a SOAP message. This structure is defined by a contract expressed in WSDL (Web Service Description Language). The exchange of messages is made possible by a communication interface consisting of: a send-message WS client and a receive-message WS service (see Section 5 for a description of a demonstrative experiment).

Interaction management. The open interaction system is managed by an intermediary that we call the *OIS manager*. The OIS manager is in charge of managing the interactions of the agents participating to the system; it is not an agent proper (in that it has no autonomy in the choice of goals or strategies), but rather a system implementing support processes. From the architectural point of view it is crucial for the openness of the entire system that the OIS is deployed on an operating environment that is completely separated form the operating environment where the interacting agents are deployed. The OIS manager interacts with all participating agents through the communication interface; its actual implementation can be distributed, to avoid performance bottlenecks. The OIS manager has the following functions:

Agent registration and de-registration. Agents are registered and de-registered as participant of the OIS, according to suitable norms specified in the system's artificial institution.

[10] http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

Management of white and yellow pages. The OIS manager maintains an updated register of all current participants. The white pages list all current participants with their agent identifiers. The yellow pages describe the services provided by the participants.

Communication. The OIS manager may also offers *services* in support of the communication layer. Those services may be activated or not on the basis of the types of the interaction that the agents registered to the OIS want to realize: a private interaction or a certified interaction. Given that in open systems it is impossible to make strong assumption on the internal structure of the interacting agents, an important service consists in dispatching all the messages through the OIS manager, and to charge it of checking messages for compliance at the levels of syntax, terminology, and preconditions. If the message is correct at all these levels the OIS manager is then in charge of forwarding messages to receivers, otherwise an error message is sent to the sender. Those checks can be performed thanks to the formal representation of the messages proposed in the previous section. Another relevant service that the OIS manager may provide consists in keeping track of the semantics of messages and in monitoring the evolution of the state of the system, for example by checking if obligations become fulfilled or apply sanctions when become violated.

5 Demonstrative Implementation

We now briefly describe a demonstrative implementation[11] of the connectivity and messages layers. The demonstrative implementation shows how OIS participants can communicate through an intermediary, which implements the low-level functions of the OIS manager introduced in Section 4. To effectively communicate, each component of the systems must be able to process OCeAN-ACL messages. More specifically, both the intermediary and the participants possess a *listening point* to receive messages and a *talking point* to send messages. The listening point is represented as a *web-service service*[12]. As such it is exposed on the Internet via a contract defined in a WDSL file. The talking point is a *web-service client*, which communicates in conformance with the previously mentioned contract. More precisely the contract defines an operation called sendmessage, which expects as input an OCeAN-ACL message and the listening point provides a service that is in charge of delivering its triggering message to its encompassing program. The service contract contains a reference to the abstract syntax of the messages proposed in this paper that is specified using an XML Schema stored in the file aclmessage.xsd. The message transfer format is SOAP,

[11] The RDF Schemas, the WSDL contract, the XML Schema of the envelope and payload of messages, the ontology, and the Codec described in this paper are available at http://www.people.lu.unisi.ch/fornaran/OCeAN-ACL.html

[12] This name is due to the fact that we consider "web-service" as the name of a technology that we differentiate from the formal concept of service and service client as understood in SOC/SOA, which in principle can be provided on the web using web-service technology or something else.

the message transfer protocol is HTTP, and the actual communication content part of the message is the serialization into a string of the content of the message written using RDF/XML triples.

To run the experiment, we have implemented the intermediary in Java and two dummy participant agents in the JADE (Java Agent DEvelopment Framework) framework. A crucial advantage of this approach is the provision of a human-readable communication contract that can be easily handled with the support of runtime frameworks coming along with web-service technology such as Apache CXF. Hence anyone can easily generate the infrastructure to handle the transmission of a message abiding to the exposed messaging protocol and adapt it to his need, in order to participate in the OIS. Effort will only be required for the handling of ACL communication content.

We will now briefly describe how the sender agent, the receiver agent and the intermediary service handle the communication content and semantics in our prototype. Regarding the sender agent, it is a software that starting from a representation of the content that has to be sent is able to serialize it in RDF/XML and send it as the content of a SOAP message. Given that in our implementation the sender is a JADE agent (but in principle it can be any software able to comply with the conventions and with the web service contract) the internal representation of the content of the message is realized according to the JADE Content Reference Model [2]. As required by the JADE Content Language Architecture this internal representation is serialized in RDF/XML by using the OCeAN RDF Codec that we have adapted to the ocean-rdf0 language. In the OCeAN RDF Codec we use the JENA Semantic Web Framework for reading and writing RDF triples.

In our demonstrative implementation the intermediary has to analyze the content of the message to check whether its content is written using the terminology specified in the language and ontology components of the message and the preconditions for the performance of a message of that type are satisfied; the intermediary also keeps track of the semantics of the message. Our intermediary does all these operations in the following way: it receives a message and, before forwarding it, it checks the content of the message by creating an internal representation of it by using again the OCeAN RDF Codec; then by using OWL-API[13] it changes the ontology's ABox and TBox.

6 Comparisons with other approaches

The contribution of this paper regards mainly: (i) the proposal of using RDF (augmented with application-independent terms) as content language of a commitment-based ACL, whose semantics is represented using an OWL 2 DL ontology; and (ii) the proposal of using service oriented technologies for realizing open systems whose components can interact using the proposed message layer.

The idea of using RDF as content language has been proposed for the first time by FIPA in 2001. The main differences between the FIPA's proposal and the

[13] http://owlapi.sourceforge.net/

one presented in this paper are: (i) in FIPA's application-independent content language, fipa-rdf0, it is impossible to represent events, which are useful to specify conditions for the performance of actions and the relationship with time; (ii) FIPA's ACL semantics is based on the agents' beliefs and intentions, and does not take into account the deontic consequences of the exchange of messages; (iii) the adoption of semantic web languages that are international standards makes it possible to realize systems able to reuse existing ontologies sometime already considered a standard (like the FOAF ontology), moreover when programming the participants and the OIS it is possible to use the numerous existing tools for programming, editing, validating, and reasoning on ontologies. In [12] a proposal of using the Darpa Agent Markup Language (DAML) language for expressing the content of message is presented, the main concepts of the content language are similar to the one presented in this paper, except for the use of variables that in RDF are not supported, but nowadays a possible solution is using SPARQL (the query language for RDF) expressions that contains variables. The main problem of the proposal of using DAML is that it is not the current standard for representing knowledge in the Semantic Web. In [9] a proposal of using OWL DL as content language of FIPA-ACL is presented and the limits of FIPA SL as content language are discussed. Differently from this paper we do not propose to use OWL DL for expressing the content of messages, this because we want to avoid to propose to use a language with an expressivity that is far from the expressivity of other languages in use in agent systems like SL or KIF. The two approaches have in common the idea to propose to use OWL for expressing the semantics of messages, even if in this paper we propose a commitment-based semantics instead of FIPA-ACL semantics.

Even if the framework and the architecture of open interaction that we have presented covers only the lower layers of an interaction, we can briefly compare our approach with other multiagent frameworks like JADE [1] and RETSINA [11]. In JADE agents have to run on the same platform and therefore must be homogenous or may run on different interconnected FIPA platforms and in this case cannot rely on a dedicated third party authority, that would be responsible of managing name conflicts, searching for agents with certain characteristics, and enforcing the conventions on the semantics of messages. The architecture of the RETSINA framework is very close to the one proposed in this paper; the point on which they differ is the choice that we made to use standard technologies for the specification of the software architecture. In particular our proposal of using Web Service technologies like WSDL, SOAP, and HTTP for message transfer is similar to the one proposed in [8], where messages are sent between JADE platforms using SOAP over HTTP, but in our proposal the content of SOAP messages is described with an XML Schema instead of using jade custom binary data format, a solution that better supports openness.

References

1. F. Bellifemine, A. Poggi, and G. Rimassa. JADE a FIPA-compliant agent framework. In *Proceedings of PAAM*, volume 99, pages 97–108, 1999.

2. G. Caire and D. Cabanillas. *JADE Tutorial: Application-Defined Content Languages and Ontologies*, last update: 15-April-2010.
3. T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
4. N. Fornara. *Specifying and Monitoring Obligations in Open Multiagent Systems using Semantic Web Technology*, volume Semantic Agent Systems: Foundations and Applications of *Studies in Computational Intelligence*, chapter 2, pages 25–46. Springer-Verlag, 2011.
5. N. Fornara and M. Colombetti. Operational specification of a commitment-based agent communication language. In *Proceedings of The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy*, pages 536–542. ACM, 2002.
6. N. Fornara and M. Colombetti. *Specifying Artificial Institutions in the Event Calculus*, volume Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models of *Information science reference*, chapter XIV, pages 335–366. IGI Global, 2009.
7. N. Fornara, F. Viganò, M. Verdicchio, and M. Colombetti. Artificial institutions: A model of institutional reality for open multiagent systems. *Artificial Intelligence and Law*, 16(1):89–105, March 2008.
8. D. Greenwood, M. Lyell, A. Mallya, and H. Suguri. SOAP based Message Transport for the Jade Multiagent Platform. In *Industry Track Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '09, pages 101–104. ACM, 2009.
9. B. Schiemann and U. Schreiber. OWL-DL as a FIPA-ACL content language. In *Proceedings of the Workshop on Formal Ontology for Communicating Agents, Malaga, Spain,*, 2006.
10. M. P. Singh. A social semantics for agent communication languages. In *Proceedings of the 1999 IJCAI Workshop on Agent Communication Languages*, volume 1916 of *Lecture Notes in Artificial Intelligence*, pages 31–45, Berlin, 2000. Springer.
11. K. Sycara, J. Giampapa, B. Langley, and M. Paolucci. Software engineering for large-scale multi-agent systems. chapter The RETSINA MAS, a case study, pages 232–250. Springer-Verlag, Berlin, Heidelberg, 2003.
12. Y. Zou, T. W. Finin, Y. Peng, A. Joshi, and R. S. Cost. Agent communication in daml world. In W. Truszkowski, C. Rouff, and M. G. Hinchey, editors, *WRAC*, volume 2564 of *Lecture Notes in Computer Science*, pages 347–354. Springer, 2002.