An Infrastructure for the Design and Development of Open Interaction Systems

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

[1] Università della Svizzera Italiana,
via G. Buffi 13, 6900 Lugano, Switzerland
{daniel.okouya,nicoletta.fornara,marco.colombetti}@usi.ch
[2] Politecnico di Milano,
Piazza Leonardo da Vinci 32, 20135 Milano, Italy
marco.colombetti@polimi.it

Abstract. We propose an infrastructure for the design and development of Open Interaction Systems (OISs), based on solutions from Service Oriented Architecture, Semantic Technologies, and Normative Multiagent Systems. OISs are open to diverse types of participants (software agents), and enable them to interact with each other to achieve their objectives. To do so the participants are allowed to interact in compliance with previously agreed-upon regulations provided by the system and on the basis of the semantics of the communicative acts performed, both of which are enforced by the system. The infrastructure we propose, based on the OCeAN metamodel of Artificial Institutions, involves four layers: (i), the Messaging Layer, which enables observable ACL message exchanges between heterogeneous participants while respecting ownership boundaries; (ii), the Core Service Layer, which enables the participants to perform observable non-communicative actions relevant to the ongoing application; (iii), the Bridging Layer, in charge of interpreting the participants' actions in a form suitable for regulation; and (iv), the Regulation Layer, which holds the regulations and enforces them with respect to the participants' activities.

Keywords: Open Interaction System, Artificial Institution, Ontology, Normative System, Agent Communication.

1 Introduction

Open Interaction Systems (OISs) are distributed systems which diverse types of participants (i.e., software agents) can freely join with the goal of interacting with each other to achieve their personal objectives. To do so the participants are allowed to interact by exchanging messages with rigorously defined syntax and semantics, in compliance with previously agreed-upon norms provided by the system; both the norms and the syntax and semantics of the communication language are enforced by the system.

In our past work we have proposed the OCeAN metamodel [13] for the specification of OISs. In this paper we describe an infrastructure, currently under development, for the actual implementation of such systems. In designing this infrastructure we aim at guaranteeing openness and interoperability, while exploiting as far as possible technologies that are sufficiently mature and stable, and are already adopted by a large industrial community. Among such technologies we include standard Service Oriented Technologies [5] and Semantic Web Technologies [15].

The infrastructure we propose involves four layers: (i), the *Messaging Layer*, which enables heterogeneous participants to interact with each other through communicative actions while respecting ownership boundaries; (ii), the *Core Service Layer*, which allows the participants to exploit the support services offered by the OIS to perform non-communicative actions; (iii), the *Bridging Layer*, in charge of interpreting the participants' actions in a form suitable for regulation; and (iv), the *Regulation Layer*, which holds the norms regulating the interactions and enforces them relative to the participants' actions. More specifically:

- The Messaging Layer provides a Messaging Protocol based on standard technologies (HTTP, SOAP, WSDL) and uses Web Service Technologies for the transfer of messages between participants, by prescribing the use of a specific message transfer service exposed via WSDL; messages realize communicative or institutional acts and comply with OCeAN-ACL [11], an Agent Communication Language based on Semantic Web Technologies, and on OWL 2 DL in particular.
- The Core Service Layer makes certain complementary services available to the participants (e.g., an OIS realizing an e-marketplace may offer services related to payment, product delivery, and so on), and thus allows them to perform observable non-communicative actions relevant to the ongoing interaction.
- The Bridging Layer interprets the participants' communicative and non-communicative actions in a form suitable for regulation; coherently with the OCeAN metamodel, such actions either result into commitments (like in the case of acts of informing, requesting, etc.) or are regarded as attempts to perform institutional actions relying on suitable *count-as* rules.
- Finally, the Regulation Layer realizes a normative context (again according to the OCeAN metamodel), that is, a set of artificial institutions specifying the institutional actions that can be performed and the set of norms that have to be followed.

In this paper we provide a detailed specification of all layers and describe the implementation, currently under development, of an infrastructure oriented to the implementation of an open e-marketplace. A graphical representation of the layered architecture is given in Figure 1; the components, the ontologies, and the relationships among the components shown in the figure will be explained in the sections describing the corresponding layers. The paper is organized as follows. In Section 2 we describe the functionalities pertaining to the Messaging

Layer and how we implement them by exploiting standard Web Service Technology. In Section 3 we briefly sketch how the core services offered by the OIS can be actually realized, considering an e-marketplace as an example. In Section 4 we describe the functionalities pertaining to the Regulation Layer and how we implement them by exploiting Semantic Web Technologies, and OWL ontologies in particular. In Section 5 we explain how relevant events taking place at either the Messaging or the Core Service Layer are made available to the Regulation Layer. In Section 6 we review some related works. Finally in Section 7 we draw some conclusions and briefly describe our plans for future work.

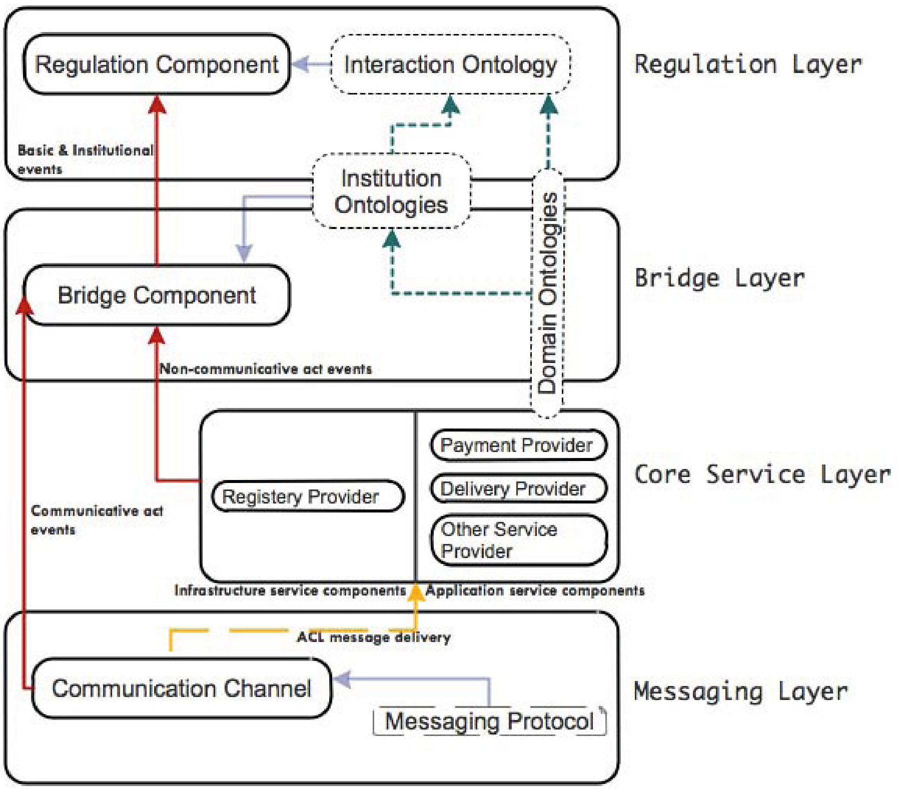

Fig. 1. An Architecture for Open Interation Systems

2 The Messaging Layer

In an OIS, a large part of the participants' interactions is carried out through the exchange of suitable messages. Therefore the bottom layer of our infrastructure provides the means to enable heterogeneous participants to interact with each other by exchanging messages in a fully interoperable fashion. In addition, it does so in such a way that it ensures the observability of these interactions, to the purpose of regulation.

To this end our infrastructure integrates principles from Service Oriented Architecture (SOA) and from Multiagent Systems (MAS). First, a message transfer approach is prescribed that is neutral to the internals of the participants, and leverages standard technologies to facilitate widespread adoption. This is in contrast with approaches based on some of the most well known ready-to-use messaging technologies like JMS[1], RMI[2], and CORBA [18], which bind either to a particular programing language [14] or to a programing language paradigm [18]. Such approaches do not fully decouple the end-point implementation from the messages, thus limiting interoperability [20,3]. Our architecture, following SOA's principles of loose coupling, solely prescribes a message format together with its transfer protocol, both of them strictly decoupled from the end-point implementation, while insisting as much as possible on the adoption of standard technologies [6,5,23].

Next, we add to the architectural prescriptions a combination of the SOA concept of a message, as comprised of carrying and content information, with the MAS idea of a powerful and flexible Agent Communication Language (ACL), to enable the participants to interact through the performance of communicative acts in a totally interoperable way. More precisely, we take the content part of a SOA message to represent the various components of a suitably designed ACL, in which the application-independent and application-dependent components are clearly distinguished.

Finally, to enable the observability of the communicative acts performed by the participants to the purpose of regulation, this layer includes a *Communication Channel* (CC) in charge of mediating message exchanges between participants. More precisely, to communicate with other registered participants, a registered participant shall send its messages to the address of the CC, with the name of the desired participants as the recipients. The CC receives the message which, if approved by the regulative process of the infrastructure, is then delivered to the intended participants. If the message is not approved, the CC rejects it and sends a suitable explanation back to the sender.

These architectural requirements are met in the infrastructure as follows. In the first place, the infrastructure provides for a messaging protocol based on standard neutral technologies: HTTP, SOAP[3], and WSDL[4]. In other words, Web Service Technology is adopted for message transfer between participants, by specifying a message transfer service, exposed via WSDL, in which HTTP is used for the transport of messages and SOAP for their structure. Our choice is motivated by the fact that this technology represents a standard approach for making available over the network functionalities that are triggered or delivered by exchanging messages.

[1] `http://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html`
[2] `http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/index.html`
[3] `http://www.w3.org/TR/soap`
[4] `http://www.w3.org/TR/wsdl`

In the second place, the infrastructure implements the messages of our ACL[5] as the body of SOAP messages. From the syntactic point of view, the ACL we propose is very close to KQML[6] and FIPA ACL[7], from which however it substantially departs as far as semantics is concerned (see Section 5). As with FIPA standards, our ACL comes with a separate Content Language (CL). Our CL is defined as an OWL Ontology, the *Content Language Ontology* [11], which plays a role similar to FIPA-RDF in FIPA ACL.

Thus, realizing the first two requirements, we define a WSDL file with only one service, which is the delivery of an ACL message, carried in the body of a SOAP message. The WSDL contract represents all message forms that can be exchanged between entities of our OIS, with the requirement that the message contains the address to reply to according to the same contract. Communication between participants is only allowed through the use of this service; consequently, all participants are required to be equipped with a suitable communication module, composed of: (i), a *listening point*, that is, a web-service provider exposing a message delivery service defined according to our WSDL contract; and (ii), a *talking point*, that is, a web-service client that requests the delivery of a message in conformance with that contract [10].

A crucial advantage of this approach is the provision of a messaging protocol in the form of a WSDL contract, which is both human readable and machine processable. Such a contract can be easily handled with the support of runtime frameworks coming along with Web Service Technology, such as Apache CXF [1,16]. We use CXF to automatically generate the core of the communication module of the participating component of our infrastructure; hence anyone can easily generate the necessary facilities to handle the transmission of messages abiding to the exposed messaging protocol and adapt it to their need, in order to participate in the OIS.

Finally, to deal with message transfer the infrastructure provides an implementation of the CC as a Java component, developed with CXF as exposed above.

3 The Core Service Layer

As we have already remarked, in an OIS a large part of the participants' interactions is carried out by exchanging suitable messages; as required by the Messaging Layer, in our infrastructure such messages are always ACL messages realizing communicative acts. However, most types of applications will also require the execution of actions that are not strictly speaking communicative. We identify these actions as *non-communicative acts* and classify them into two categories: first, non-communicative acts concerning the interaction between the participants and certain components of the infrastructure, designed to provide support to the participants' activities (as we shall see, these non-communicative acts are typically

[5] `http://www.people.lu.unisi.ch/okouyad/AclOverSoapHttpMP.wsdl`
[6] `http://www.csee.umbc.edu/csee/research/kqml/`
[7] `http://www.fipa.org/repository/aclspecs.html`

application independent); second, application-specific non-communicative acts concerning the interaction between participants.

More specifically, on the one hand some of the application-independent non-communicative acts are intended to support the enforcement of ownership boundaries between participants, enabling them to connect with each other without introducing dependencies. For example, the infrastructure provides for a Registry component (see below), through which the participants can register or deregister by performing suitable actions. Although the registration and deregistration processes do require the performance of certain communicative acts (more precisely, of the request to be registered or deregistered), the actions of registering or deregistering a participant are not themselves communicative; rather, they are non-communicative actions made available to the participants by the infrastructure, through the provision of services that may be invoked using communicative acts (requests). On the other hand, some of the application-specific activities (i.e., some of the activities that are carried out between the participants) may require the performance of application-specific non-communicative actions which, as in the case of communicative acts, must be made observable to the infrastructure. In an e-marketplace system, for example, when engaging in a purchasing activity, after settling a contract by performing suitable communicative acts, the buyer may be required to carry out a payment, while the seller may be required to deliver a product. Both of these are non-communicative actions inherent to the purchasing activity, and as such must also be visible to the infrastructure.

To sum up, the objective of this layer is to equip the infrastructure so that: (i), it enables the participants with performing all the infrastructure-specific non-communicative actions belonging to the direct interactions between the participants and the infrastructure itself; and (ii), it can observe the performance of the application-specific non-communicative actions inherent to some of the activities occurring between the participants. In order to achieve these goals our infrastructure requires that the participant register to the IOS; to this end it provides a Registry component, implemented in Java, to serve as a White Pages Service. The Registry provides, among others, for the *registering* and *deregistering* actions; as this component is endowed with ACL-processing capabilities, the participants can request its services using ACL messages.

In unison with the approach used for the communicative acts (i.e., that the actions occurring between the participants are mediated by the infrastructure), the infrastructure can also mediate the non-communicative actions that are application-specific. In this respect, however, the Core Service Layer proceeds differently than the Messaging Layer. Indeed, the different communicative actions that can be performed by the participants are the same across applications; thus, the observation process necessary to handle them is also application-independent, and therefore can be achieved by a generic component (the Communication Channel). In contrast, non-communicative acts occurring between participants are typically application dependent: their presence, what they achieve, and how they achieve it, always depend on the application being realized. Therefore unlike communicative acts that are always available to the participants, the presence of non-communicative actions is application-specific; for instance, the availability of a shipping action could be irrelevant for certain applications, such as an e-market for computational services. Moreover, when present the performance of non-communicative acts can substantially vary depending on the requirements of the applicatios in which they are performed. In the case of a payment, for instance, while one application may require a system like PayPal, another one may require a direct bank-to-bank transfer or a cheque payment, which would need to go through different steps and to supply different information. Another important difference is that, unlike communicative acts, non-communicative actions can also vary in nature, that is, they can be electronic, physical, or involve both aspects.

Hence, to mediate non-communicative actions the infrastructure must take into account their fundamental application-oriented features, as well as the fact that they can involve any combination of physical and electronic aspects. To achieve this, our architecture prescribes that the Core Service Layer provides for the incorporation of observable application-specific components, offering to the participants specific services of mediation for application-specific, non-communicative actions. These components must be such that they seemingly interoperate with the participants for the invocation of the actions that they mediate, whose performances must be observable by the infrastructure.

To this purpose, on the one hand this layer specifies the interfaces of the mediating components, so that the relevant parts of the infrastructure can take into account the performance of the non-communicative actions they are in charge of; on the other hand, it prescribes the characteristics that the components must posses so that their services can be seemingly consumed. In support of that latter point, the layer mandates the use of communicative acts to invoke the mediation services; that is, theses services are invoked using ACL messages, which brings the advantage of providing a unique invocation protocol, independently of the nature and level of complexity of the services. Of course, this implies that the mediating components must be able to process certain ACL messages. This contrasts with the message-transfer mediation service provided by the Communication Channel, which is invoked using a SOAP message (as a typical web-service).

Meanwhile, it is important to remark that our infrastructure does not require that the mediating components directly perform the non-communicative actions they supply: indeed they may do so, or guarantee that they are performed by certain external systems, or simply acknowledge their external realization when so informed by a group of participants who have agreed to exploit an external service. In this regard, the layer classifies theses services into two distinct categories: *internal services* and *external services.* In the former case, the service is internally managed by the component itself; this means that when requested by a participant, the component takes charge of the execution of the activities involved in the service. In the latter case, which represent a very decentralized approach providing more freedom to the participants, the execution is guaranteed by the participants themselves, which then inform the infrastructure of the results. Here mediation plays the role of a neutral authority that acknowledges the realization of services taking place out of its direct control, according to the specific rules governing the application.

4 The Regulation Layer

Once heterogeneous participants, possibly belonging to different owners, can interact with each other as exposed above, it is necessary that they get provided with some form of harnessing framework defining norms that regulate their interactions. This is particularly important as it allows the participants to have reasonable expectations with respect to the interactions they engage in order to achieve their objectives. Moreover given that we target systems as e-marketplaces, taking in account the sensitive nature of their activities, the architecture prescribes the realization of a neutral third-party component in charge of analyzing the participants' interactions (by using the information received by the Bridging Layer as described in Section 5), with the aim of monitoring the evolution of the interaction and specifying and enforcing the norms of the regulating context.

To realize all these functionalities we introduce in the proposed architecture the *Regulation Layer*. This is based on the OCeAN metamodel [12], in which regulating contexts are defined as *artificial institutions* that provide a high-level representation of a specific set of institutional actions together with the norms that govern them, and of the institutional objects that need to be observed to monitor the evolution of the state of the interactions. For every specific application, such institutions are operationalized by grounding them in the current domain [13,8].

The Regulation Layer must possess a formal representation of the state of the interaction suitable to carry out automated reasoning. In particular this representation has to include specifications of: (i), the regulating context in force; (ii), the types of events and actions the application is dealing with; (iii), the application-independent and application-dependent knowledge defining the relevant objects and their states during the interaction; and (iv), the instances of the institutional actions and events that actually take place in the system. Reasoning will then allow the system to monitor the evolution of the state of the interaction, detecting in particular norm fulfillments and violations.

Our infrastructure meets these requirements in the following way. We define our regulating context as an OCeAN artificial institution. The first regulating context we have operationalized so far is the *Commitment Institution*, which regulates agent interactions in terms of the commitments they make to each other by performing communicative acts [8,9]. This is an application-independent foundational institution, from which more specific application-dependent institutions (like for example the institutions formalizing different types of auctions) can be defined. The Commitment Institution specifies commitments as institutional objects, together with their life-cycle rules and the institutional actions that allow an agent to create, cancel, or otherwise manipulate them; this enables us to monitor the state of an interaction in term of the evolution of the commitments that the participants make to each other. Application-dependent regulating contexts (like for example those relevant to e-commerce) are also represented as OCeAN institutions.

In our infrastructure, institutions as well as domain entities (e.g., the products that are exchanged in the e-market) are represented by ontologies specified in OWL 2 DL [15], the standard language for defining ontologies in the Semantic Web. Also the state of an interaction is represented in an OWL ontology, that we call the *Interaction Ontology*, which is continually updated while the interaction proceeds (see Section 5). More precisely, the Interaction Ontology contains a representation of the institutional objects defined by the institutions in force, together with the institutional actions that can create and manipulate them. To serve this purpose, the Interaction Ontology imports:

– an *Upper Ontology* specifying common application-independent concepts like *agent*, *action*, *event*, and *object*;
– the *SWRL Temporal Ontology*[8] used for representing and reasoning on instants and intervals of time;
– the OWL ontologies used for representing the relevant artificial institutions, like in particular the *Commitment Institution Ontology*[9];
– the *Domain Ontology* used for representing relevant domain knowledge.

Some of these ontologies are described in details in [11]. The ontology imports are realized according to an architecture [11] that we have crafted specifically to avoid conflicts and duplications of the application-independent concepts (like agent, action, temporal interval, etc.) on which several ontologies overlap.

Using OWL 2 DL reasoning, our representation makes it possible to monitor the state of the interactions according to the rules of the context. Thus, equipped with it, in compliance with the prescriptions of the architecture which require a neutral third-party component to enact this functionality, our infrastructure provides for a regulation component which plays the role of interaction manager, in charge of monitoring regulations and requesting their enforcement when necessary. To this purpose, the regulation component relies on the Pellet OWL 2 reasoner[10], used in conjunction with the OWL-API[11]. When started with the paths to the relevant ontologies as parameters, it loads them and creates the initial Interaction Ontology. Then, a suitable assertion is added to the ABox and the reasoning process is triggered every time a relevant event happens, such as the elapsing of a pertinent instant of time, or the realization of an institutional or non-institutional action or event. As we shall see below (Section 5), suitable representations of relevant actions and events are provided by the Bridging Layer.

Implementing interaction monitoring by OWL 2 DL reasoning is not straightforward. First, as participants interactions have to be represented over time, it is

[8] `http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalOntology`
[9] `http://www.people.lu.unisi.ch/okouyad/CommitmentOntology.owl`
[10] `http://clarkparsia.com/pellet/`
[11] `http://owlapi.sourceforge.net/`

necessary to carry out some kind of temporal reasoning. For instance, if a participant is committed to another agent to realize a given action before a deadline, in order to deduce that after the deadline the commitment is either fulfilled or violated it is necessary to deduce that the deadline has elapsed. This cannot be specified by OWL axioms alone; therefore, SWRL[12] rules containing temporal built-ins have been added to perform suitable temporal inferences. Such rules exploit the SWRL Temporal Ontology developed by the Protege group [17], which provides a time representation format that is suitable for calculation, is aligned with the current XSD standards, and defines a rich set of temporal builts-ins that can be used to extend our OWL ontologies with SWRL rules. However, given that these built-ins are not SWRL standards, they are not natively supported by reasoning engines; as the Protege group has provided an implementation for reasoning with these built-ins only with the Jess rule engine, we have developed our implementation for extending the reasoning capabilities of the Pellet reasoner by using the custom built-ins definition mechanism provided with it.

Representing the evolution of the state of interactions (including for example the new commitments that the participants bring about), by means of a continuous update of the Interaction Ontology at run-time [9], is a delicate task because it may introduce inconsistencies. More specificaly, in our formalization of the Commitment Institution Ontology[9] (presented in [7] with the name *Obligation ontology*), we specify that an *actioncommitment* (i.e., a commitment to perform an action, intuitively equivalent to an obligation), has an associated temporal interval, within which the action must be executed. Determining this interval can involve several steps depending on the properties inherited by the commitment at its creation. In certain situations, such as when the action-commitment is conditional, it only becomes activated if a specific triggering event or action takes place; when this activation occurs, the beginning and the end instants of the interval associated to the action-commitment have to be set. For example, if the exchange of a message commits a participant to deliver a product within two days, on condition that the receiver of the product performs a payment, then the action-commitment will be created as soon as the message is exchanged, but will only be activated when the payment takes place. At activation time the interval will be determined as follows: (i), its beginning is set at the time instant of the activation; and (ii), its end is set at the beginning plus two days. In principle, all this may be expressed by a suitable SWRL rule. However, if several actions belonging to the activation class of the obligation take place, the SWRL rule will be activated several times and the interval of the obligation will be represented incorrectly. It turned out that this problem cannot be solved inside the OWL ontology, even if additional SWRL rules are used; therefore we regulate the activation of the relevant SWRL rule with an external Java program that exploits the OWL-API to check that an interval that is already set is not further changed. In short, some reasoning steps and calculations have to be made outside of the reasoner, in order to properly manage the Interaction Ontology.

[12] `http://www.w3.org/Submission/SWRL/`

5 The Bridging Layer

To regulate the interactions it is necessary to capture the participants' actions and other relevant events that take place in the system, and represent them in a form that suits the abstraction level at which regulation operates. This is the purpose of the Bridging Layer (or Bridge, for short). This layer, which shares with the Regulation Layer the definition of the institutions in force, operates as detailed in the sequel.

First, all events (inclusive of the participants' actions) that are relevant for regulation must be observed by the Bridge. These events take place either at the Messaging Layer or at the Core Service Layer. As far as the former is concerned, the relevant events consist in exchanges of ACL messages, which are made available for observation by the CC (Communication Channel) component of the Messaging Layer. To the purpose of regulation, it is crucial that all message exchanges between participants take place through the CC provided by the infrastructure. As we have already remarked, however, message exchanges are not the only events that need regulation. Among these also certain non-communicative events are included, like for example the actions of paying or delivering a product. These events are made observable by the Core Service Layer.

The observed events have to be represented in a form that is suitable for regulation. In particular, given that the Regulation Layer relies on artificial institutions, representing an actual observed event in a form suitable for regulation involves producing a representation that is compatible with the specification of the artificial institution.

In the OCeAN metamodel, artificial institutions deal with two types of events, that we respectively call *basic* and *institutional events*. An institutional event Y is an event that is brought about by the performance of a lower level event X, thanks to suitable *counts-as* rules, provided that certain enabling conditions C hold. For example, an artificial institution may specify that a certain type of message sent by a suitably empowered agent A will count as an institutional action of opening an auction. Contrastingly, a basic event is an event that can be directly produced by a participant, without the need of realizing it through the performance of another, lower level event. For example, performing the concrete action of sending a message to another participant is represented in the institution as a basic event of message exchange.

Transforming an observed concrete event in a form suitable for regulation requires producing a representation of either a basic or institutional event. In the Regulation Layer, both artificial institutions and the concrete domains over which they operate are specified as OWL ontologies. Thus the infrastructure transforms the actual observed event into OWL individuals that belong to classes of events pertaining either to the institution ontologies or to the domain ontologies. More accurately, as institutional events are always grounded on basic events, this transformation process consists of: (i), creating an OWL individual representing the basic event; and (ii), creating an OWL individual representing the institutional event, if this is dictated by a count-as rule belonging to the institution in force.

As specified by the OCeAN metamodel, we provide a set of application-independent *counts-as* rules that associate to message exchanges (considered as basic events) the creation of suitable commitments (considered as institutional events): these rules are part of the Commitment Institutions and specify the application-independent component of the semantics of OCeAN-ACL. In general, according to the semantics of OCeAN-ACL the exchange of a message is interpreted as an attempt to perform an institutional action of commitment manipulation, which is precisely specified by a $counts-as$ rule; such an attempt will be successful if, and only if, the enabling conditions C associated to the rules hold. For example, the exchanges of commissive messages (like promises) and of directive messages (like requests) are interpreted in the Commitment Institution as attempts to perform institutional actions that create *action commitments* [22], that is, commitments to perform the action described in the content part of the message. Commitments of this type can be considered as equivalent to *obligations*; for example, if agent A promises to agent B to pay a given sum of money M for a given product P, the communicative act will be interpreted as an attempt to create an obligation of agent A to pay M euros to B for product P[13]. When the Bridging Layer delivers this institutional action to the Regulation Layer, the Interaction Ontology will be updated with a new institutional object of type *Obligation*, with A as the debtor, B as the creditor, and the payment of M euros for P as the content. Thereafter, the obligation will be monitored for its fulfillment, violation or cancellation as part of the process of interaction monitoring carried out by the Regulation Layer. Requests are treated in a similar way, except that they involve one more step; more precisely, a request is interpreted as the attempt to create an *action precommitment* (or *preobligation*), which in turn leads to an attempt to create an obligation for the receiver, if the receiver accepts the request (i.e., the preobligation).

Assertive communicative acts (like the acts of informing) are conceptually different from commissives and directives, because they introduce *propositional commitments* [22], which cannot be interpreted as ordinary obligations. For example, if agent A informs agent B that the product delivered is damaged, this commits A to the truth of what is said (i.e., that the product is indeed damaged), but does not immediately obligate A to perform any predefined action (in particular, of course, it does not obligate A to damage the product). We have not yet worked out a representation of propositional commitments for our infrastructure: this issue is therefore deferred to future works.

Finally, there is another type of communicative acts, which following the terminology of Searle's Speech Act Theory [21] we call *declarations*; examples are declaring that an auction is open, or that a specific agent is the winner of an

[13] Note that a message exchange, considered as an attempt to perform an institutional action, is successful only if the enabling conditions associated to the relevant $counts-as$ rule hold; for example, as specified by the OCeAN metamodel, a message stating that a commitment is cancelled will be successful if it is sent by the creditor of the commitment to its debtor, while it will fail to achieve the cancellation if it is sent by the debtor to the creditor.

auction run. Declarations are carried out by exchanging suitable ACL messages, with *declaration* as the performative, and a content that represents the institutional action whose performanceis being attempted. Coherently with the OCeAN metamodel, such messages are interpreted within an artificial institution through a *counts-as* rule, which generates the declared institutional action provided that certain enabling conditions hold. Typically, a condition for the successful performance of a declaration is that the actor has the *institutional power* to perform the declared institutional action (e.g., only an auctioneer can possibly open an auction). Such institutional powers are associated at design time to the different roles that can be played by a participant in an institution, and are checked at runtime by the Regulation Layer.

In practice, to achieve this transformation from basic events to institutional events, the OWL specifications of application-independent concepts (such as agent, action, event, object, time instant, time interval, etc.) are shared between the Content Language Ontology (see Section 2), the relevant Institution Ontologies, and the Domain Ontologies over which the ongoing application operates and on which the institutions are grounded. Sharing is achieved thanks to the ontological architecture introduced in the Regulation Layer, which eliminates all the ontological mapping hurdles that would have otherwise been necessary to handle for the full transformation process to take place. Indeed it allows to seemingly go from one representation to another; for instance, going from the communicative action $promise(A, B, pay(book01, 5))$ (which involves the Content Language Ontology and a concrete Domain Ontology) to the institutional action $create - obligation(A, B, pay(A, B, book01, 5), instant01)$ (which involves the Commitment Institution Ontology and the same Domain Ontology) is achieved smoothly thanks to the underlying shared concepts of agent, action, and object. If these concepts were not shared appropriately, mappings would have been necessary between the specifications of these concepts in different ontologies. The same principle applies, for example, when a non-communicative action of payment takes place: the actual action is represented by the OWL individual $pay01$ of class Pay, suitably related with individuals A as its actor, B as its recipient, $book01$ of class $Book$ as its object, 5 as its amount in euros, and $instant01$ as its instant of performance; this individual can imply the institutional event $tranfer - ownership(B, A, book01, instant01)$ of a hypothetical Ownership Institution (where the target representation is understood as B transferring the ownership of $book01$ to A at $instant01$).

In sum, to perform this bridging process so as to update the regulation component, the Bridge is launched with the paths to all the relevant ontologies (that it loads using the the OWL-API), and a reference to the regulation component. The process is then triggered each time it receives updates from the the Communication Channel or a Core Service component.

6 Related Work

Among the recent multiagent infrastructures focused on OISs, which in particular share the aim of providing the regulation of the participants' interactions in the form of a neutral third-party functionality as part of the overall support that they deliver, the Magentix2 Open multi-agent systems platform[14][4] represents the state of the art on the matter. In particular it is the most advanced operational infrastructure, which includes many of the recent advances in the OIS area. Interestingly, we happen to share strong architectural similarities. We therefore start by exhaustively comparing it with our infrastructure. Then we will provide another comparison with a promising infrastructure currently under development, 2COOM, which exemplifies the rising trend of environment-based MAS infrastructures.

At a very abstract level our infrastructure and Magentix2 share the same architectural approach. More precisely, although their respective layered architecture are slightly differently structured, they present the same abstract organisation: a top part concerned with regulation specification and management, a bottom part concerned with the support of observable interactions between heterogeneous participants, and a middle part concerned with the monitoring of the participants' interactions according to the rules in force and their enforcement when deemed appropriate. Consequently, differences only appear in the way the parts are concretely realized, with the most fundamental of them occurring in the middle part. This reflects a common vision of the role of the infrastructure, but divergences on how its different parts may concretely operate to achieve it.

More specifically, at the top level, Magentix2 adopts the metamodel of virtual organizations, which specifies roles with norms including platform generic roles such as OMS (Organization Management System) and DF (Directory Facilitator), for the specification of a regulation structure. Our infrastructure also defines a regulation structure at this level, but one that is based on the OCeAN metamodel of artificial institutions (see Section 4). While a thorough comparison of the two metamodels is outside the scope of this paper, it can be safely said that both infrastructure intend to provide similar regulating structures, which in particular are centered on non-regimented norms, to harness the participants' activities.

At the bottom level, both infrastructures provide an observable vehicle for the participants to interact with each other. To that end, they use similar approaches, but differ in the general understanding of interactions. Indeed the OCeAN metamodel classifies actions into communicative and non-communicative ones, which Magentix2 does not, in that it only considers communicative actions. Consequently, while we divide the bottom part of the infrastructure into two layers (Messaging and Core Service), with the upper one devoted to non-communicative actions and the lower one devoted to communicative ones, Magentix2 only provides one interaction level which corresponds to our lower layer.

As far as communicative interactions are concerned, the two infrastructures operate in a similar manner (as they both provide an end-point neutral messaging protocol with a broker for interoperable communication between heterogenous participants), but diverge in the choice of the technology. Where we use Web Service Technology (SOAP, HTTP, WSDL) with the SOAP body structure

[14] `http://www.gti-ia.upv.es/sma/tools/magentix2/`

defined as an OCeAN-ACL message for messages exchange, Magentix2 adopts the Advanced Message Queuing Protocol (AMQP)[15], with the message body structure defined as a FIPA-ACL message. The use of Web Service Technology is more widespread and therefore we expect its adoption to be less problematic than that of AMQP.

As previously mentioned, the sharpest differences between the infrastructures occur in the middle part, whose functionalities can be summarized as follows: (i), observing actual events such as message exchanges or core-service events; (ii), representing observed events in a form suitable for regulation; (iii), checking them against the regulations for monitoring purposes; and (iv), enforcing the relevant regulations when deemed appropriate. It is with (ii) and (iii) that the two infrastructures differ substantially.

With our infrastructure, checking against regulations is done by means of reasoning over a representation of the state of the interaction, carried out within an OWL ontology that includes the institutions in force and the norms coming along. While our norms and their instantiations (in terms of obligations and prohibitions) are represented as OWL individuals, their activation, cancellation, fulfillment and violation conditions are represented as event types (i.e., as subclasses of class Event). With this approach we can use the full power of DL reasoning to match the representations of actual events and actions with the conditions and contents of norms. This process is much more powerful than the one adopted by Magentix2, which relies on the matching of a restricted subset of first-order logic formulas.

A further important difference between Magentix2 and our infrastructure is that the latter does not rely on an application-independent semantics of ACL messages. In our infrastructure, based on the OCeAN metamodel, the application-independent part of messages (i.e., all the components of an ACL message with the exception of its content) is given a uniform semantics across applications. Moreover, such semantics allows for a representation of messages (produced by the Bridging Layer) that immediately relates message exchanges to the Regulation Layer. This means that only application-specific non-communicative events will need to receive a special treatment in different applications of the infrastructure. Conversely, Magentix2 does not provide for any application-independent connection between the participants' actions and regulation, thus making the conversion to different application more expensive and error prone.

Another relevant infrastructure for OISs currently under development is 2COMM [2] which, similarly to ours, firmly relies on the principle of artificial institutions to structure interactions. 2COMM is mainly build on top of the CArtAgO framework [19], which is based on the Agents & Artifacts metamodel, and to a lesser extent on the JADE infrastructure. In its essence, 2COMM proposes to use the programmable artifacts of CArtAgO to provide for a mediated communication between participants and to model the institutional framework. More specifically, an *artifact* provides for a set of operations, which in the case of 2COMM are communicative actions, and it also manages the institutional inter-

[15] `http://www.amqp.org/`

pretation and monitoring of those actions in terms of operations on commitments; this is the reason why those artifacts are called *commitment-based communication artifacts*. The set of available actions, together with the roles to which they belong and their institutional interpretation, constitute what 2COMM calls a *commitment protocol*. 2COMM provides for an abstract *BasicCommitmentCommunicationArtifact*, defining agent available operations such as *enacting* or *deacting* a role, as well as the internal operations to manage commitments (*create*, *realize*, and so on). Then, through an inheritance process, a designer can define every specific commitment-based communication artifact protocol that will be available to the participants, like for example the *Contract Net Protocol communication artifact*; the inheritance process consists in adding specific pairs of public operations-internal commitment operations grouped by roles. 2COMM also provides for the necessary management infrastructure, thus enabling agents to use the protocols in a coherent fashion by means of the *ArtifactManager* Jade Agent, which communicates with the requesting agents via FIPA-ACL messages (as provided by Jade).

Although the infrastructure proposed in this paper and the 2COMM one share similar intents, they are sharply different and they significantly diverge in the way we use institutions to harness the interactions, but also in some specific supports that our infrastructure provides. These differences can be delineated as follows. First, at the lowest level, while we provide *intrinsic interoperability* as a support to openness, 2COMM, due to its dependence on the CArtAgO and the JADE infrastructure, does not. Indeed, on one hand, initiating the CArtAgO services, entering the workspaces where the artifacts are situated, as well as programming or using such artifacts, can only be done through the use of Java code. For instance, artifact operations must be implemented as Java methods, called using introspection through a Java API made available by the CArtAgO framework. Therefore, given that the agents that interact with a CArtAgO Environment should be developed using Java, the resulting infrastructure is not completely interoperable with agents developed using other programming languages. We think that this is an important aspect in the realization of OIS in the domain of electronic marketplace, which in essence tries to reach out to as many participants as possible. Indeed even if our infrastructure is developed in Java, its usage does not prescribe agents developed in Java or somehow using it, nor does its customization to a specific application (i.e., the development of new service providers for the core service layer). For no participating component (i.e., core service provider or OIS participant) we make any assumption on how they should be realized, and simply require that they abide to our interoperable messaging protocol.

At a higher level, the differences could be articulated around the following points: (i), how we represent and monitor institutions; (ii), how we use institutions, in particular in the light of the commitment-based semantics of communicative acts; (iii), the set of mechanisms for mediating the interactions at run-time; and (iv), the type of communication.

A fundamental aspect that differentiates our infrastructure is on the approach used for the representation and monitoring of institutions. As previously discussed also in the comparison with Magentix2, we use OWL 2 DL to represent institutions. For instance, we model commitment as OWL individuals, with their contents and conditions modeled as OWL classes. This makes it possible that actions or events that are not known in details in advance, but are simply described by means of a class, will fulfill certain commitments. More generally, we use OWL 2 DL to reason on the evolution of the state of the interaction. Differently, 2COMM uses Java objects to represent both the commitments and the other facts that are relevant to manage the evolution of the overall state of the institution. In particular, a fact has a string field to represent the name of a predicate it represents, and an argument field that is a list of objects. This representation implies a matching process to check whether the content or the condition of a commitment is satisfied (a combination of syntactic string and java object matching). Moreover our model allows to express and manage commitment deadlines, which allows us to detect violations, an aspect that is not tackled in the 2COMM approach.

A second difference is on how institutions are used for the specification of the semantics of communicative acts. Indeed, we make a clear distinction between the application-independent *Basic Institution* (i.e., our definition of an application-independent commitment semantics for OCeAN-ACL) which specifies no norms, and the special-purpose institutions (e.g., auction, ownership) which are fully fledged normative institutions. 2COMM through the use of commitment protocols (i.e., operationalized institution) seems to mix the semantics of communicative acts with the normative aspect of fully fledged institutions. This may have the negative effect of nullifying the advantages of defining a commitment-based semantic of communicative acts. In fact instead of having agents able to freely choose the communicative acts to perform, the 2COMM approach guides the course of message exchange. For instance, an initiator in a *Contract-Net protocol* can only perform the *call for proposals* act and it cannot perform any other act that is not in its role. We believe that the normative dimension should be reserved to aspects that are exclusive to the special-purpose institutions. For instance, in an auction only the auctioneer has the power to open an auction. However, if in the meantime a participant performs a communicative act (e.g., an inform, request, promise, or call for proposal), its institutional effects should be retained. In other words, we want to let the agents free to explore any course of message exchange that they see fit to reach their objectives.

Thanks to the use of the CArtAgO environment, the 2COMM framework provides an efficient checking of the powers for performing institutional actions and the runtime mechanisms for letting the agents to perceive the state of the interaction. Our infrastructure has yet to provide for it, we plan to realize it as part of the ongoing build-up of the support we are providing for the special-purpose institutions. Another difference is on how communication is conducted in the two infrastructures. On one hand, while we both claim to mediate communicative actions, we actually operate differently. Our infrastructure forward ACL messages between participants. It records the actions if necessary, and most importantly their institutional effects. Differently 2COMM does not transfer messages. Actually, messages are Java method call on the artifact, which modifies its state depending on certain conditions. The possible modification can be: the record of the fact that a method was called, the fact that a communicative act was performed, or its institutional effects. Then, it is that change that will be observed by the participating agents. A final observation is that the proposed infrastructure is more agile, because it firmly separates concerns (messaging, core service, bridge, regulation) whereas the 2COMM infrastructure do not, as it combines everything in an artifact. Moreover, by using method calls 2COMM loses the flexibility gained in separating the various components of an agent communication language (i.e., ACL syntax, Content Language, and Domain Ontologies).

We can conclude that, aside from the interoperability issues discussed above, only empirical studies will reveal whether one of the two approaches, or perhaps a mixture of the two, is better and in which domain.

7 Conclusions

In this paper we have presented an infrastructure for Open Interaction Systems, based on the OCeAN meta-model and currently under implementation. Our main concerns in the development of the infrastructure are, on the one hand to guarantee openness and interoperability, and on the other hand to rely as much as possible on technologies that are sufficiently mature and stable, like Service Oriented and Semantic Web Technologies, to facilitate adoption by the industry.

The infrastructure has been divided into components to separate different concerns, which brings several advantages: on the one side, it enables us to distribute the infrastructure and to use techniques of dynamic adaptation (such as cloning and self-deletion) to manage overhead issues; on the other side it enables us to provide targeted upgrades and developments of the infrastructure. So far, for prototyping purposes the infrastructure is being implemented as a monolithic multi-threaded Java application; nevertheless, the different components are present and well separated so that they could be easily extracted to provide a fully distributed infrastructure.

In the near future we intend to complete the implementation and test of the prototype. In particular we plan to complete the formalization in OWL of the semantics of the various type of communicative acts, to separate the various component of the prototype, and to test it with the formalization and execution of an e-marketplace, inclusive of the OWL ontologies representing the relevant institutions and domain knowledge.

References

1. Balani, N., Hathi, R.: Apache CXF Web Service Development. Packt Publishing (2009)
2. Baldoni, M., Baroglio, C., Capuzzimati, F.: 2COMM: a commitment-based MAS architecture. In: Cossentino, M., El Fallah Seghrouchni, A., Winikoff, M. (eds.) EMAS 2013. LNCS (LNAI), vol. 8245, pp. 38–57. Springer, Heidelberg (2013)
3. Chiarabini, L.: CORBA vs. Web Services (May 2004), `http://www.itu.dk/~oladjones/mastersthesis/materialsfromportals/corbaversuswebservices.pdf` (accessed March 14, 2013)
4. Criado, N., Argente, E., Noriega, P., Botti, V.: MaNEA: A Distributed Architecture for Enforcing Norms in Open MAS. Engineering Applications of Artificial Intelligence 26(1), 76–95 (2012)
5. Erl, T.: Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall (August 2005)
6. Erl, T.: SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl). Prentice Hall PTR, Upper Saddle River (2007)
7. Fornara, N.: Specifying and Monitoring Obligations in Open Multiagent Systems Using Semantic Web Technology. In: Elçi, A., Koné, M.T., Orgun, M.A. (eds.) Semantic Agent Systems. SCI, vol. 344, pp. 25–45. Springer, Heidelberg (2011)
8. Fornara, N., Colombetti, M.: Specifying Artificial Institutions in the Event Calculus. In: Dignum, V. (ed.) Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models, ch. XIV. Information Science Reference, pp. 335–366. IGI Global (2009)
9. Fornara, N., Colombetti, M.: Representation and monitoring of commitments and norms using OWL. In: AI Communications - European Workshop on Multi-Agent Systems (EUMAS) 2009, vol. 23(4), pp. 341–356 (2010)
10. Fornara, N., Okouya, D., Colombetti, M.: A Framework of Open Interactions based on Web Services and Semantic Web Technologies. In: Proceedings of the 9th European Workshop on Multi-Agent Systems, EUMAS 2011 (2011)
11. Fornara, N., Okouya, D., Colombetti, M.: Using OWL 2 DL for Expressing ACL Content and Semantics. In: Cossentino, M., Kaisers, M., Tuyls, K., Weiss, G. (eds.) EUMAS 2011. LNCS, vol. 7541, pp. 97–113. Springer, Heidelberg (2012)
12. Fornara, N., Viganò, F., Colombetti, M.: Agent communication and artificial institutions. Autonomous Agents and Multi-Agent Systems 14(2), 121–142 (2007)
13. Fornara, N., Viganò, F., Verdicchio, M., Colombetti, M.: Artificial institutions: a model of institutional reality for open multiagent systems. Artif. Intell. Law 16(1), 89–105 (2008), doi:10.1007/s10506-007-9055-z
14. Hapner, M., Burridge, R., Sharma, R., Fialli, J., Stout, K.: Java Message Service Specification Version 1.1. Sun Microsystems, Inc. (April 2002)
15. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
16. Kent, T.K.: Developing Web Services with Apache CXF and Axis2, 3rd edn. Lulu.com (2010)
17. O'Connor, M.J., Das, A.K.: A Method for Representing and Querying Temporal Information in OWL. In: Fred, A., Filipe, J., Gamboa, H. (eds.) BIOSTEC 2010. CCIS, vol. 127, pp. 97–110. Springer, Heidelberg (2011)

18. OMG. The Common Object Request Broker: Architecture and Specification. The Object Management Group, pp. 1–712 (November 1999)
19. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective. Autonomous Agents and Multi-Agent Systems 23(2), 158–192 (2011)
20. Scordino, C.: How Web Services relate to the well established CORBA Middleware (April 2004), `http://retis.sssup.it/~scordino/documents/corba.pdf` (accessed March 14, 2013)
21. Searle, J.R.: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, Cambridge (1969)
22. Walton, D.N., Krabbe, E.C.: Commitment in Dialogue: Basic concept of interpersonal reasoning. State University of New York Press, Albany (1995)
23. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR, Upper Saddle River (2005)