Specifying Interaction Space Components in a FIPA-ACL Interaction Framework

Ernesto German, Leonid Sheremetov

Mexican Petroleum Institute
Eje Central Lazaro Cardenas 152, San Bartolo Atepehuacan,
Distrito Federal, Mexico
{egerman, sher}@imp.mx

Abstract. Despite the acceptance of FIPA-ACL as a standard for agent communications, there exist a gap between its specification and infrastructures to support interactions among agents. The hypothesis we study in this paper is that interaction space components must be specified and described in depth by taking into account an explicit six-layered FIPA-ACL communication model. Based on this model, generic components can be described for a FIPA-ACL interaction framework. An implementation for interaction components is given within the CAPNET agent platform in an integrated way with the agent interaction architecture. Working with interaction space capabilities for engineering agent interactions is pointed out through a MAS example.

Keywords: FIPA-ACL, Interaction Framework, Interaction Space.

1 Introduction

Beyond dealing with communication issues at the level of data and physical message transport, Agent Communication Languages (ACL) are complex structures composed of different sublanguages that specify the message content syntax and meaning, interpretation parameters such as the sender and receiver, and the pragmatics of the intention of the message. Furthermore, next to message exchange, interaction also includes interpretation and validation that the message has been correctly interpreted.

In spite of many efforts on designing flexible and robust agent interactions, very little attention has been paid so far on providing support for runtime processing such interactions using higher level concepts than messages. Indeed, current Multi-Agent System (MAS) infrastructures (such as languages, toolkits, frameworks and platforms) are limited to simple message sending and receiving for processing agent interactions [1]. Although interaction protocol is a higher level concept than messages, they are supported at runtime only for controlling the sequence of messages but not for processing the whole set of activities involved in ACL interaction. Nevertheless, the increasing complexity of MAS integration requires more effective interactive behaviors based on message semantics and pragmatics [2], [3].

Though FIPA-ACL communication language has become a standard to engineer agent-to-agent interactions, two main objectives persecuted by this language, autonomy and interoperability, are not addressed by MAS engineering. Our experience in developing MAS with current FIPA-ACL infrastructures tells that interactions typically have been engineered using somewhat ad-hoc and developer-private communication assumptions made for reasons of communication efficiency or developer convenience [4], [5]. Furthermore, knowledge of these assumptions is critical to properly interpret and understand messages at runtime, becoming autonomy and interoperability almost impossible to achieve [6]. So, while application specific agents are useful to test and validate different approaches to develop agents, they can be extremely difficult to generalize and extend without extensive interaction with the original developers.

The paper focuses on the problem of interaction by explicitly arranging and engineering different layers found in the FIPA-ACL language specification[1]. These layers go from physical transport and encoding messages to internal agent processing for syntax, semantics and pragmatics of messages. In particular, we think that explicit support for capabilities helps to fill the gap between FIPA-ACL specification and implementation/runtime interaction processing. To facilitate the engineering of MAS interactions we developed interaction space components as an important step to address the problem of interaction. Our approach considers that these interaction components can be integrated in an agent interaction architecture.

The structure of the paper is the following. In section 2 the specification of an explicit FIPA-ACL communication model is given. Section 3 describes the generic interaction space components required to fulfill computation at each layer of the model. Section 4 gives details of the interaction space components implemented into the CAPNET agent platform. In section 5 an example shows how the interaction space components can be created in applications and explains how components are used at runtime by the agent interaction architecture. Finally, some related works are discussed and conclusions of our work are given.

2 FIPA-ACL Communication Model

Till now, the FIPA model has focused more on how agents could communicate by separately specifying different components. However, little work has been done on explicitly specifying organization and integration of these components to enable message processing by agents. The paper considers a six-layered FIPA-ACL model which is inspired in a recent revised FIPA-ACL model [7]. The approach based on layers is taken to better organize and build the interaction components from an engineering perspective because it lets specify not only interaction components but also computation in the context of runtime message validation process [8]. The focus of the paper is to describe the components needed at each layer by means of development tools implemented in our agent interaction framework and not to discuss in depth this communication model. In this section we briefly describe the FIPA-ACL communication model and in section 3 the interaction space components are detailed.

[1] Foundation for Intelligent Physical Agents. FIPA Communicative Act Library Specification http://www.fipa.org/specs/fipa00037/ and FIPA ACL Message Structure Specification http://www.fipa.org/specs/fipa00061/

In figure 1, the connection between FIPA-ACL communication model and the application layer of the OSI reference model is shown. The FIPA-ACL model starts on top of the OSI reference model [9] extending the application layer to support six FIPA-ACL computation layers: Message Transport, Message Encoding, Content Expression Syntax, Content Expression Semantics, Communicative Acts and Interaction Protocols. The unit of processing at each layer is a FIPA-ACL message.

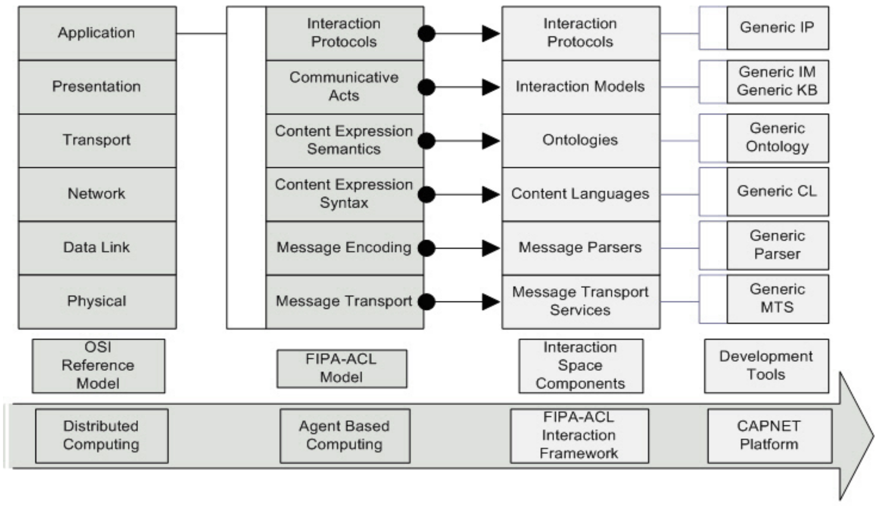

Fig. 1. FIPA-ACL communication model: an engineering perspective

At the message transport layer, agents look for and use asynchronous message transport protocols to interchange messages through a physical network. The next layer validates message structure and encoding because agents serialize messages through the network. Messages are encoded using data structures such as XML, string formats and bit efficient schemas, further than binary codifications. Furthermore, message information such as agent identifiers, type of message and payload require parsers. Content expression syntax is a layer where agents recognize the entities built-in into the content of messages by determining whether the content structure is correct in accordance with a common content language representation. The fourth layer refers to the use of ontologies to describe the meaning of content by explicitly representing domain concepts. At the layer of communicative acts agents have to manage the messages taking into account the pragmatics of each type of communicative act. Almost all communicative acts entail access to a knowledge base where application domain information is stored. When communications use a pre-defined sequence of messages among involved agents, interaction protocols components are used.

3 Interaction Space Components

In our previous work a FIPA-ACL interaction framework was described through three main notions: interaction space, interaction models and interaction architecture [8].

The Interaction Space (IS) is an environment that stores interaction capabilities of the agent that can be accessed in order to validate interactions at runtime. It is integrated by the following components: message transport services, message parsers, content languages, ontologies, interaction models, interaction protocols and a knowledge base. An Interaction Model (IM) represents a modular unit permitting the validation of a simple interaction. IM includes five modules for programming validation of syntax and semantics content, feasibility preconditions, rational effect of messages and interaction termination. The Agent Interaction Architecture (AIA) is defined as a component to control creation and processing of interactions through validation of interaction models within the interaction space of an agent. Interaction space is part of agent architecture. The main objective of this framework is to propose components for extending FIPA-ACL interaction infrastructure by means of interaction space capabilities.

3.1 Message Transport Service

The first layer is composed of message transport services (TS). TSs are components that agents use to exploit several available network infrastructures. Concrete networking technologies are available along distributed computing infrastructures so that different TSs could be implemented for exploiting those advantages of each type of technology such as SOAP-XML for web services-based agents, HTTP for web-based agents and TCP for remote object-oriented agents. Since some of these communication services are commonly used in known MAS infrastructures, in this framework they are considered to be part of the interaction space like specific components that will be invoked dynamically when agents need their services.

Although each TS has its own implementation features they can be implemented using common interfaces in order to be added to the interaction space of the agent. Basically, this type of services must give functionality to process the sending and receiving transport messages. Transport message is the unit of communication in this layer of the model and includes message payload and envelope information.

3.2 Message Parser

At the layer of message encoding, each message is either encoded or decoded[2] by a component called a message parser. The main activity of parsers is to find out whether the structure of the message complies with FIPA-ACL. This is a first level of syntactic validation of the message. Typically, several message parsers can be available as software components. These parsers must be implemented following a well defined interface to generate and parse messages represented through different codification schemas such as XML and others types of strings. The information about parser components is explicitly available in the interaction space. With this information, agent interaction architecture can dynamically analyze the message requirements applying the specific parser.

[2] Encode means what is usually called “generate” or “format” and decode is similar to parse.

3.3 Content Language

Agent communication is designed to represent the content of messages following certain common criteria in such way that content can be understood by both sides of the communication. Since agents could manage different content languages (usually written by different programmers), similar basic elements should be used. Based on the specification of each communicative act of FIPA-ACL, five types of entities that can be part of message's content are implicitly found: actions, propositions, domain objects, references to objects and FIPA-ACL messages (table 1).

Table 1. Content requirements for FIPA-ACL communicative acts

Communicative act	**Content entities**
accept-proposal, agree, cfp, failure, propose, refuse, reject-proposal, request-when, request-whenever	action, proposition
request, cancel	action
confirm, disconfirm, inform, inform-if, query-if	proposition
inform-ref	object reference
not-understood	action, message, proposition
propagate, proxy	object reference, message, proposition
query-ref, subscribe	object reference

We define a Generic Content Language component with these set of basic content entities. The GCL also contains a set of content objects to build message content combining one or more basic entities. Every concrete CL should give only one object per communicative act. For example the "*request-when*" communicative act combines an action and a proposition in the content. Since every entity and content object is designed to be used in a serialized way in messages, they must give two functions: the first one is used to serialize the entity in an encoded format in such way that it can be part of the message. The second one does the opposite task: from a serialized representation gets the entity information and re-builds the entity as a memory object. The validation criteria for each concrete CL are left free to CL programmers because they depend on the particular requirements of each type of entity.

3.4 Ontology

The communication model of FIPA-ACL is based on the assumption that two agents try to interact sharing a common ontology of the domain to give meaning to the entities represented in a message's content. For a given domain, agents can decide to access ontologies explicitly represented and stored. In this paper, we consider that ontologies engineering must share common design lines. That is why, we propose to define a Generic Ontology (GO) as software component. Based on GO, concrete ontologies can be built and added as part of the interaction space of agents and can be accessed to validate the semantics of message's content at runtime.

The GO is basically formed by all content entities given by generic content language but messages. GO has two parts: in the first one, collections for actions, propositions, domain objects and object references store the information about the domain. In the second part there are a set of validation functions for each type of content entity forming the ontology and one validation function for each type of communicative act. The criteria for internal organizing, storing and validating the entities in the ontology are left open for developers of concrete implementations.

3.5 Interaction Model

An interaction model is a key concept of the framework for implementing the communicative acts layer in the FIPA-ACL model. An IM is seen as an interaction component for validating single-message interactions. For each communicative act, the IM is composed of the modules covering five validation phases: validation of the content structure with a specific content language, validation of content semantics with a specific ontology, validation of feasibility preconditions, validation of rational effect, and validation of the termination conditions of the interaction.

Depending on the interaction requirements of each agent, different interaction models implemented according with supported interaction capabilities such as communicative acts, content languages and ontologies are required. Each IM is going to be stored in the interaction space so it could be automatically used when messages fit its requirements. The idea is that IMs can be as reusable for different application agents as possible or at least ready to be refined by specializing functionality.

When agents interact and try to achieve pragmatics of communicative acts (feasibility preconditions and rational effects), almost always they have to store, query or modify concrete information about the application. The knowledge base is an interaction space component very important to complete several types of interactions.

Being consistent with the knowledge model given by the FIPA-ACL followed in both the generic CL and generic ontology, the KB must give the possibility of managing actions, propositions and domain objects in order to allow agents to reason about requirements of the communicative acts. For example, when a request message is going to be sent or is being received, the agent has to decide whether the action is known. That is why we consider that actions must be part of the knowledge base. Regardless any concrete implementation of the KB, this software component must give functions to add, query and remove actions, propositions and domain objects.

3.6 Interaction Protocol

The framework requires interaction protocols to attend interactions composed of more than one message. To build concrete IPs, we propose to define a generic interaction protocol as a component with IP common attributes. The specification for a Generic Interaction Protocol (GIP) is given by a unique name of IP, a name of the content language, a name of the ontology and the implementation engaged of controlling the execution sequence and states of the IP. Each agent will be able to know the set of protocols it can use at runtime when interactions occur because they will be stored in the interaction space. How IPs are invoked and executed is a matter of agent interaction architecture and it is out of the scope of this paper.

4 CAPNET Interaction Space

The current version 2.0 of the CAPNET agent platform [10] is empowered with the interaction framework described in this paper. In the CAPNET, each type of interaction component is implemented following an object oriented design. The *InteractionSpace* class is a container of concrete objects representing capabilities that can be used dynamically by the validation process of the AIA. Each concrete component has its own unique descriptive information so that message attributes can be used to resolve at runtime invocation of the correct component, depending on the communication requirements. Each interaction capability is engineered by following interfaces and base classes that represent generic component functionality. In this way, capabilities can be implemented by reusing and extending from them exploiting the runtime polymorphic advantages for checking and resolving types.

4.1 Message Transport Services

To help the messaging system to work dynamically (and eventually to make agents more autonomous) CAPNET transport services are implemented by following the *IGenericTransportService* interface (figure 2-a). This interface defines methods for sending (*sendMessage*) and receiving (*receiveMessage*) messages. The base class *BaseTransportService* declares attributes for transport service type (*MTSType*) and address (*address*).

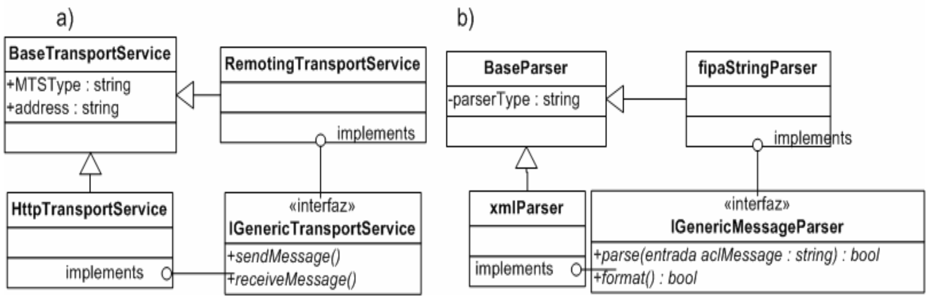

Fig. 2. CAPNET Message Transport Services (a) and Parsers (b)

At the moment, we have implemented two concrete message transport services. In the first one (*RemotingTransportService*), we configure a TCP connection by using distributed .NET framework remote objects for intranet agent applications. The second service (*HttpTransportService*) is an HTTP server infrastructure based on request-response connections to send and receive messages beyond local area networks and for web based agent applications. In the interaction space, we add one single instance of each TS to make them available at runtime.

4.2 Message Parsers

In the CAPNET implementation, the *IGenericMessageParser* interface describes the generic functionality for parser components. Two methods are described to cover the parsing of messages: on the side of the sender agent, *format* should be used for converting a message to its textual representation ready to be communicated by a transport service. *Parse* is the method for checking message syntax and for recovering the message information from a textual representation when a message is received on the side of the receiver agent. As it is shown in figure 2-b, the basic class *BaseParser* is given to be extended by concrete classes like *xmlParser* and *fipaStringParser*. While the first one serializes messages by using XML formats and conventions the second represents messages as string format.

4.3 Content Languages

CAPNET CLs design is based on the Generic Content Language specification and implemented by the *GenericCL* class (see figure 3-a). *GenericCL* class has a *CLName* attribute to assign a unique identifier of the CL. Also this class is composed of a set of basic entity classes (explained in section 3.3) that implements the *ISerialization* interface supporting methods for serialization syntax validation (*validateDescription*) and for converting the entity to a serializable format (*setDescription*).

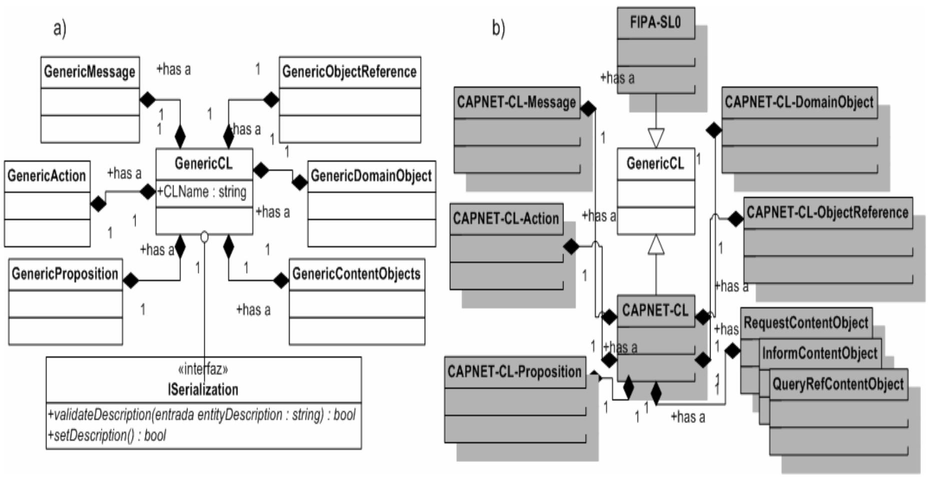

Fig. 3. CAPNET Content Language implementation

Following these design rules, we have developed two concrete CL classes in CAPNET (figure 3-b). *CAPNET-CL* [11] is a proprietary language based on FIPA-RDF0 to represent the syntax of entities. *FIPA-SL0* is the implementation of CAPNET to support FIPA-SL0 specification. Both CLs inherit from *GenericCL* class and implement each entity by the *ISerialization* interface.

4.4 Ontologies

For engineering ontologies, CAPNET offers the *GenericOntology* class which is composed by several common ontology attributes (figure 4). Ontologies must have a unique name for identifying them in the interaction space (*OntoName*). The attribute *CLName* is the name of the content language the entities managed by the ontology belong to. As it was established in the Generic Ontology, this software component contains collections for storing actions, propositions, domain objects and references to domain objects (*ActionsSet*, *PropositionsSet*, *DomainObjectsSet* and *ObjectReferencesSet* respectively).

Concrete ontologies must implement the *IOntology* interface to offer common functionality. This interface has functions to search entities (*searchAction*, *searchProposition*, and so on) and to add entities (*addAction*, *addProposition*, and so on). Finally, the interface includes methods to validate the content object of each type of communicative act supported by the ontology (*validateInform*, *validateRequest*, *validateQueryRef*, and so on). We have developed the *CAPNETOntology* concrete class by using the *CAPNET-CL* entities.

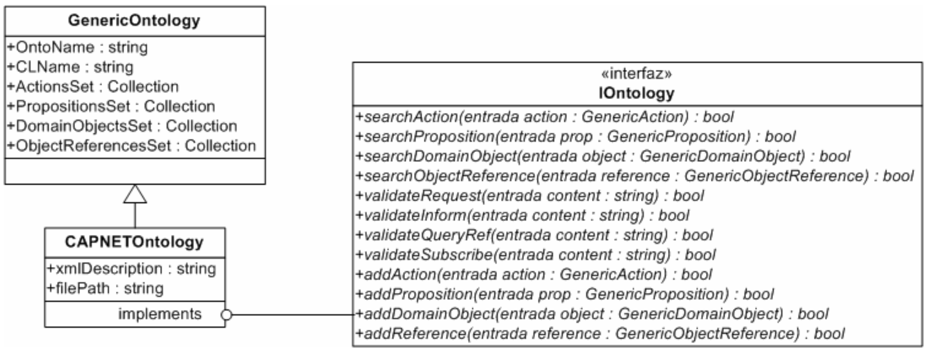
GenericOntology
+OntoName : string
+CLName : string
+ActionsSet : Collection
+PropositionsSet : Collection
+DomainObjectsSet : Collection
+ObjectReferencesSet : Collection
CAPNETOntology
+xmlDescription : string
+filePath : string
implements
«interfaz»
IOntology
+searchAction(entrada action : GenericAction) : bool
+searchProposition(entrada prop : GenericProposition) : bool
+searchDomainObject(entrada object : GenericDomainObject) : bool
+searchObjectReference(entrada reference : GenericObjectReference) : bool
+validateRequest(entrada content : string) : bool
+validateInform(entrada content : string) : bool
+validateQueryRef(entrada content : string) : bool
+validateSubscribe(entrada content : string) : bool
+addAction(entrada action : GenericAction) : bool
+addProposition(entrada prop : GenericProposition) : bool
+addDomainObject(entrada object : GenericDomainObject) : bool
+addReference(entrada reference : GenericObjectReference) : bool

Fig. 4. Design of ontologies in CAPNET

4.5 Interaction Models

Interaction Models are software components based on the *GenericInteractionModel* CAPNET class. Some of them are illustrated in figure 5-a. When a message is going to be processed, the AIA looks for an IM that fits the message requirements.

To control the execution of concurrent IMs at runtime, a set of common attributes identified for interaction models were implemented. *InteractionId* is a unique number to identify each individual interaction. *IMName* is used to identify the communicative act of the interaction. The field *message* indicates what message the IM object is related to at runtime. *CLName* and *OntologyName* attributes store the names of the content language and ontology used to represent and validate syntax and semantics of the entities included in the message content. The same type of communicative act can be implemented by different interaction models combining different content languages and ontologies because an agent can participate in different application domains. Each IM developed for a specific agent must implement the *IInteractionModel* interface where the five phases of IM validation are defined (*validateCL*, *validateOntology*, *validateFP*, *validateRE* and *validateTermination*).

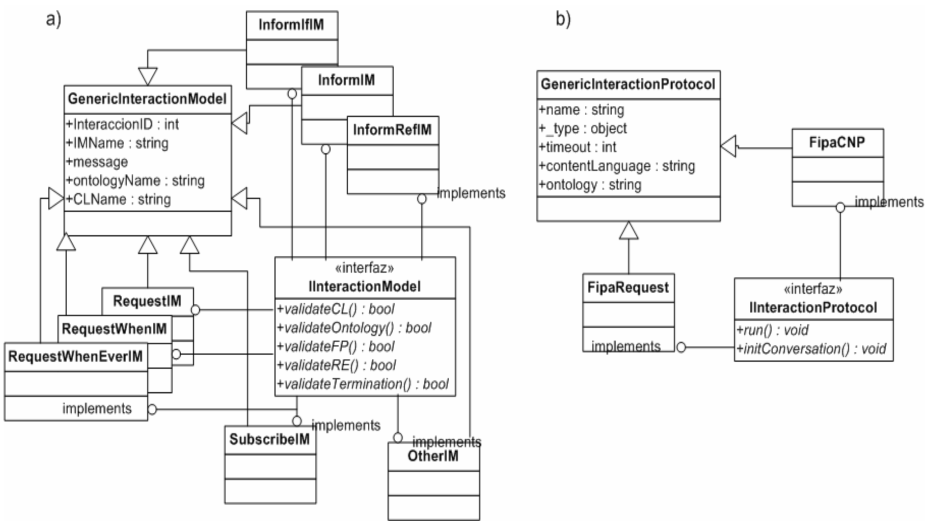

Fig. 5. CAPNET Interaction models (a) and protocols (b)

4.6 Knowledge Base

The CAPNET KB was implemented by the *KnowledgeBase* class. This component is formed by collections to store actions, propositions and domain objects derived from generic classes. Moreover, in our implementation we have found two special types of collections to temporally store monitors. Monitors are propositions and domain objects to be monitored at runtime and are useful to implement some interaction models like *request-when*, *request-when-ever*, *subscribe*, *inform-if* and *query-if* making the agent be aware when entities have changed their attributes.

Although the purpose of having a knowledge base in the agent interaction space can help the agent to do many internal reasoning activities, in this paper we focus on the necessary functionality to carry out interactions. In this sense, the knowledge base class has methods to *add*, *search* and *remove* specific entities and methods to *add*, *update* and *remove* entity monitors when interactions take place at runtime.

4.7 Interaction Protocols

Each interaction protocol is implemented as a software component that includes necessary attributes to dynamically determine its execution. We propose the *GenericInteractionProtocol* base class to encapsulate such attributes that agent interaction architecture can read when messages require the use of a specific IP.

In the case of the CAPNET IP development, the base class considers that each IP has a unique name (such as FIPA-REQUEST, FIPA-CNP, and so on), a timeout (a configurable amount of time the agent is going to wait for the next message in the sequence), content language (the name of the CL used by the protocol) and the ontology (the name of the ontology) as it is shown in figure 5-b. Concrete IP classes inherit from the base class and also should implement the *IInteractionProtocol* interface where *run* and *initConversation* methods are defined to allow agents concurrently execute several IPs. *Run* is used to create a new thread of execution when the IP is instantiated in the agent interaction architecture. *InitConversation* implements the real strategy of controlling the sequence of messages.

5 Example

In this section we show a simple example that illustrates the use of the main interaction components in the context of the proposed framework. It shows how a particular agent interaction space is constructed (instantiated) as part of the agent interaction architecture. A complete description of the interaction architecture and the validation process can be found in [8].

The example is oriented to manage supplies logistics for offshore oil platforms [5]. To make supplies operations, boats and ships are required for transportation. These transportation services are offered by third party providers that the MAS have to find out and request for. This scenario configures an open and flexible environment where heterogeneous agents should interact by using different interaction capabilities. The MAS is composed by several oil platform agents that request specific supplies to a supplier agent. Supplier agent receives requests, looks for the requested supplies, and carries out the operations to negotiate the marine transport services offered by transport agents. To engineer the interactions we have built agents by using the interaction space capabilities of the CAPNET platform. Table 2 shows the interaction components incorporated to each type of agent. The code of the example is Visual Basic .NET compatible with the .NET framework 1.1.

Table 2. Interaction Space components

Agents	**TSs**	**Parsers**	**CLs**	**Ontologies**	**IMs**	**IPs**
Supplier	http TCP-Remoting	XML FIPA-string	CAPNET CL FIPA SL0	transport supply	request requestWhen inform	
Platform	TCP-remoting	XML FIPA-string	CAPNET CL FIPA SL0	supply	request inform	FIPA-request
Transport	http	XML FIPA-string	CAPNET CL FIPA SL0	transport supply	requestWhen inform	FIPA-request

The supplier agent has two transport services (*http* and *TCP-Remoting*) because it needs to interact with *Platform* and *Transport* agents. To communicate with *Platform* agents it uses *TCP-Remoting* transport service and to communicate with *Transport* agents it uses *http*. In the prototype, agents use *XML* and *fipa-string* parsers to validate syntax of FIPA-ACL messages. Resolving which parser will be used is a task done dynamically by the agent interaction architecture by checking the message field encoding for every message and by invoking the required parser component. TSs and parsers are stored in collections of the interaction space (*ts* and *p* in code 4).

When created, agents get object instances of the *CAPNETCL* and *FIPASL0* classes and store them in the interaction space (code 1). When agents are interacting with CAPNET management agents (Agent Management System and Directory Facilitator), they apply *FIPASL0* content in the messages and when communicates with supply specific agents they interchange messages codified in *CAPNETCL*.

Code 1. The content languages instances are created

```
capnetcl = New CAPNETCL(CONTENT_LANGUAGE_CAPNET-CL)
sl0 = New FIPASL0(CONTENT_LANGUAGE_SL0)
cl = New Hashtable
cl.Add(capnetcl.CLName, capnetcl)
cl.Add(sl0.CLName, sl0)
```

The MAS works with *transport* and *supply* domain ontologies. The *Supplier* and *Transport* agents load both ontologies. *Platform* agents only need *supply* ontology (code 2). Internally, these ontologies are based on the *CAPNETOntology* class and uses *CAPNETCL* to represent concrete entities. The *supply* ontology defines the *planningSupply* action to allow *Platform* agents request supplies to *Supplier* agent. This ontology also declares domain object references that can be considered as valid supplies in this domain. In the *transport* ontology the *transportSupplies* action also is defined to negotiate transport services among *Supplier* and *Transport* agents. Every action is also stored in the KB in order to be executed at runtime (code 4).

Code 2. Segment of the ontology supply creation

```
ontSupp = New CAPNETOntology("supply", capnetcl.CLName)
PlanSupplyAct = New
CAPNETCL.RDF0Action("planningSupply")
PlanSupplyAct.setact("planningSupply")
PlanSupplyAct.setactor("SupplyAgent")
ontSupp.AddAction(PlanSupplyAct)
Dim d1, d2, d11 As CAPNETCL.RDFObjectRef
d1 = New CAPNETCL.RDFObjectRef ("PERF-WATER", "No")
ontSupp.AddObjectReference(d1)
d2 = New CAPNETCL.RDFObjectRef ("DRINK-WATER", "No")
ontSupp.AddObjectReference(d2)
d11 = New CAPNETCL.RDFObjectRef("BUMP-A", "No")
ontSupp.AddObjectReference(d11)
Dim o As New Hashtable
o.Add(ontSupp.OName + ontSupp.CLName, ontSupp)
```

The agents require several interaction models in order to execute specific communicative acts supported by the MAS (code 3). For example, *Platform* agents use *requestIM* for requesting the action *planningSupply* to *Supplier* agent (and whatever supported action). Typically when they request the action, they receive the answer by an inform message that is managed by an *InformIM*. This interaction could also be carried out by applying the *fipa-request* interaction protocol when synchronous communication is preferred. In other interactions, *Supplier* agent uses *requestWhenIM* to ask *Transport* agents to execute the action *transportSupplies* for some supplies assigned to it only when the required supplies are ready to be transported. When each interaction component and collection is created, the agent programmer must create the *InteractionSpace* instance as it is shown in code 4.

Code 3. Creation of Interaction Models

```
requestWhenIM = New RequestWhenIM(ACL_REQUEST_WHEN)
requestIM = New RequestIM(ACL_REQUEST)
InformIM = New InformIM(ACL_INFORM)
Dim IM As New Hashtable
IM.Add(requestIM.IMName + capnetcl.CLName +
ontoSupp.OName, requestIM)
IM.Add(requestWhenIM.IMName + capnetcl.CLName +
ontoSupp.OName, requestWhenIM)
IM.Add(IIM.IMName + capnetcl.CLName + ontoSupp.OName,
InformIM)
```

Code 4. Creation of the Interaction Space of agents

```
Dim actions As New Hashtable
actions.Add(PlanSupplyAct.Name, PlanSupplyAct)
Dim kb As New KBManager(Props, Objects, actions)
iSpace As New InteractionSpace(ts,p,kb,cl,o,IPs,IM)
```

Figure 6 illustrates the validation process of the agent interaction architecture for the scenario of the *request* message implemented by *Platform* agents to request the *planningSupply* action to *Supplier* agent. The purpose is to show how components are instantiated and invoked by using the proposed generic design at runtime.

Messaging gets the *TCP-Remoting* TS to send the message on the side of the *Platform* agent because so is indicated in the message by the programmer. The *Supplier* agent is connected to a *TCP-Remoting* TS so it can receive the message. The messaging engine gets the *XML* parser to validate the structure of this message because the message is encoded with XML syntax. After that, the validation is made by an instance of the *requestIM* class because it has been implemented in accordance with the *CAPNET CL* content language and the *supply* ontology. Every phase of the IM is executed by the corresponding engines in the validation process. The *CAPNET CL* component is cloned from the interaction space and used to validate the *requestContentObject* by invoking its *validateDescription* method. The meaning of the content is validated when the ontology engine gets a copy of the *supply* ontology component from the IS. Then, the method *validateRequest* is invoked where the requested action is validated as part of the ontology. Finally, the requested action is searched in the KB where the capability is implemented by an executable action.

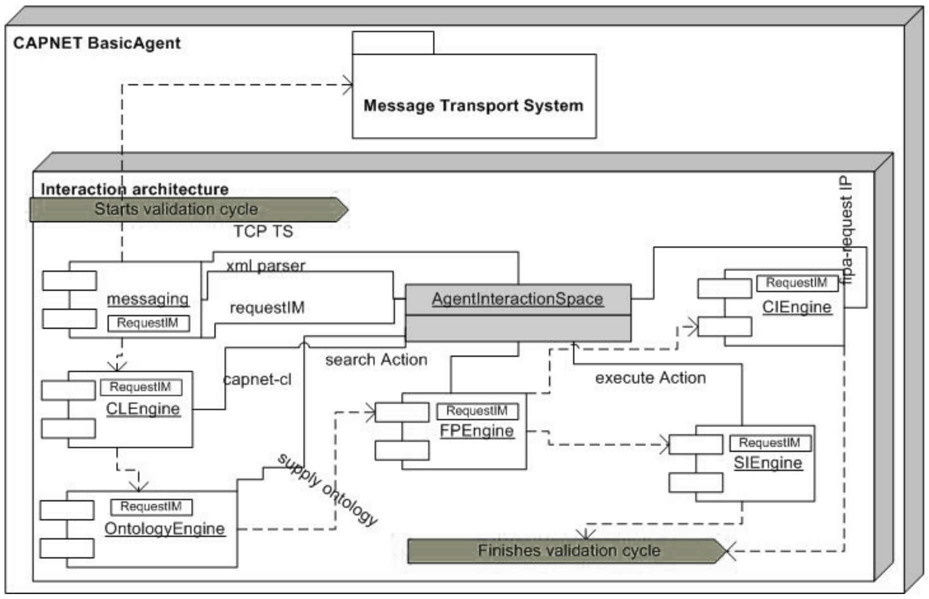

Fig. 6. Agent accessing its interaction space

6 Related Work and Discussion

Research work in agent technology is focused on moving away from the hand-crafted agents to the agents able to participate in particular institutional space enabling them to determine capabilities at runtime [12]. In such institutions, communicative interactions take place in open interaction frameworks and exist only thanks to common agreements on the basis of a shared set of conventions [13]. Nevertheless, there is little effort to model, design, and implement crosscutting agent interaction concerns which depend largely on the ability of software engineering techniques and methods to support the explicit separation of concerns throughout the design and implementation stages [14].

In the literature, there are also reported communication layered approaches like the efficient agent communication on wireless environments presented in [15] and the communication model based on interactions, conversations and ontologies described in [16]. The latter covers some specific issues but not all that we have considered in this paper.

Concerning presented approach, we briefly emphasize three issues i) how autonomy is improved, ii) what type of interoperability is enabled and iii) interaction engineering concerns.

Our autonomy's approach is oriented to process interactions. Agents are able to determine by themselves whether or not they can process unforeseen messages at runtime depending on their own interaction capabilities. This is achieved by having both explicitly represented interaction components and an inter-built agent interaction architecture. It is fairly different from that of representative works like Jadex and Jason presented in [17], which employ a reasoning architecture for deducing agent’s actions from internal domain model but not for processing ACL interactions.

Interoperability refers to the programmer’s ability to take into account at design time the interaction capabilities of the agents in order to reduce interaction among software developers. It permits development of agent interactions using common interaction space components. Other level of interoperability could be reached when agents developed within different agent infrastructures try to interoperate using the same interaction components. To reach this level of interoperability we need other platforms implement interaction components following the proposed generic components. Then experiments could be provided to test this issue in practice. Our work is different than other similar approaches found in the literature [18] [19] because it provides interoperability for each layer of the communication model.

Finally, the use of interaction space components releases developers from writing bulk of code to validate each stage of communication. Agent interaction architecture is given once by the basic agent and it takes the control of agent interaction processing. Without it, development of interactions would require writing code to control each scenario of message processing and for each agent in the MAS. That technique of programming is inflexible, repetitive and prone error because validation of messages at each layer is completely duty of the developer. As an outcome, we promote the separation of concerns by reusing, extending and sharing different interaction components.

7 Conclusions

In this paper we pointed out the interaction space components that are required to make the FIPA-ACL interaction framework carry out message processing at runtime. We organized the FIPA-ACL communication model through six layers to accomplish agent interactions: transport services, message parsers, content languages, ontologies, communicative acts and interaction protocols. Based on this model, interaction components were identified as part of each layer and arranged as core components of our FIPA-ACL interaction framework.

We proposed that every interaction component should be stored into the agent interaction space as software components that could be accessed at runtime by the agent architecture. Interaction components were defined as generic software components in order to specify their basic functionality in accordance with the expected activities they have to manage at each layer of the communication model. These interaction components are implemented within the CAPNET agent platform. We show by example of the MAS for offshore oil platform logistics how the interaction components can be created in CAPNET agents. We are convinced that proposed agent interaction architecture improves autonomy, interoperability and interaction engineering of complex multiagent systems.

Acknowledgments. The first author would like to thank CONACYT and the IMP for supporting the Ph. D. studies that originated this research.

References

1. Winikoff, M.: Implementing Commitment-Based Interaction. International Conference on Autonomous Agent and Multi-Agent Systems (AAMAS 2007), Hawaii, May 2007
2. Omicini, A., Ossowski, S., Ricci, A.: Coordination Infrastructures in the Engineering of Multiagent Systems. Methodologies and Software Engineering for Agent Systems – An AgentLink Perspective, Bergenti, F., Gleizes, M., Zambonelli, F. (Eds), Kluwer (2004)
3. Serrano, J. M., Ossowski, S.: On the Impact of Agent Communication Languages on the Implementation of Agent Systems. LNCS, Vol. 3191, Springer Verlag, (2004), 92-106
4. Smirnov A., N. Chilov, T. Levashova, L. Sheremetov, M. Contreras. Ontology-Driven Intelligent Service for Configuration Support in Networked Organizations. J. of Knowledge and Information Systems, Springer Verlag, 12(2): (2007) 229-253
5. L. B. Sheremetov, M. Contreras, C. Valencia, Intelligent Multi-Agent Support for the Contingency Management System. J. of Expert Systems with Applications, Pergamon Press, 26(1): (2004) 57-71
6. Chaib-Draa, B., Dignum, F. Trends in Agent Communication Language. Computational Intelligence, Volume 18, Num. 2, (2002) 89-1015
7. Poslad, S. Review of FIPA Specifications, IEEE FIPA Revision of FIPA Specifications Group, Foundation for intelligent Physical Agents http://www.fipa.org , September 2006
8. German, E., Sheremetov, L.: An Agent Framework for Processing FIPA-ACL Messages Based on Interaction Models. Proceeding of the eight Workshop on Agent Oriented Software Engineering (AOSE 2007), (2007) 75-89
9. Zimmermann, H.: OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnections. IEEE Transactions on Communications, vol. 28, no. 4, (1980) 425-432
10. Contreras, M., Germán, E., Chi, M., Sheremetov, L.: Design and Implementation of a FIPA Compliant Agent Platform in .NET. J. of Object Technology, vol. 3, no. 9, (2004) 5-28
11. Sheremetov, L., Batyrshin, I., Filatov, D., Martínez-Muñoz, J.: An Uncertainty Model for Diagnostic Expert System Based on Fuzzy Algebras of Strict Monotonic Operations. Lecture Notes in Computer Science, Vol. 4293 Springer Verlag, (2006) 165-175
12. Dignum, F., Dignum, V., Thangarajah, J., Padgham, L., Winikoff, M. Open Agent Systems? Agent Oriented Software Engineering (AOSE 2007), (2007) 45-59
13. Fornara, N., Vigano, F., Colombetti, M.: Agent Communication and Institutions Reality. Agent Communication, State of the Art Survey, In van Eijk, R., Huget, M., Dignum, F.(Eds) LNAI Volume 3396, (2004) 1-17
14. Garcia, A., Chavez, C., Choren, R.: Enhancing Agent-Oriented Models with Aspects. International Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2006), Japan, May 2006
15. Helin, H., Laukkanen, M.: Efficient Agent Communication in Wireless Environments. Software Agent-based Applications, Platforms and Development Kits. In Unland, R., Klusch, M., Calisti, M. (Eds), Birkhauser ISBN 3764373474, (2005) 307-330
16. van Aart, C.: Organizational Principles for Multi-Agent Architectures. Birkhauser ISBN 3764372133, (2005) 139-176
17. Bordini, R.; Dastani, M.; Dix, J.; El Fallah Seghrouchni, A. (Eds.): Programming Multi-Agent Systems. Kluwer Academic Publishers, (2005)
18. Pasha, M., Faroog-Ahmad, H., Ali, A., Suguri, H.: Semantic Grid Interoperability Between OWL and FIPA SL. Agent Computing and Multi-Agent Systems, Zhong-Zhi Shi, Ramakoti Sadananda (Eds.): LNCS 4088, ISBN 3-540-36707-1, (2006) 714-720
19. Suguri, H., Kodama, E., Miyazaki, M.: Assuring Interoperability in Heterogeneous, Autonomous and Decentralized Multi-Agent Systems. Proceedings of 6th International Symposium on Autonomous Decentralized Systems (ISADS 2003), IEEE Computer Society, ISBN 0-7695-1876-1, (2003) 17-24