JaCa-MM: A User-centric BDI Multiagent Communication Framework Applied for Negotiating and Scheduling Multi-participant Events

A Jason/Cartago Extension Framework for Diary Scheduling Events Permitting a Hybrid Combination of Multimodal Devices based on a Microservices Architecture

Juan Luis López Herrera and Homero Vladimir Rios Figueroa
Research Center on Artificial Intelliigence, Universidad Veracruzana,
Sebastian Camacho 5. Col. Centro, Xalapa, Veracruz, Mexico

Keywords: Diary Scheduling, Negotiation, Multiplatform Architecture, BDI Multiagent, Multimodal, User Interface, SOA, Microservices.

Abstract: In this research work, we present a novel BDI (Belief-Desire-Intention) multiagent software architecture for registering and scheduling multi-participant events under an automatic and semi-automatic negotiating process in a BDI multiagent context. Interactions between users and software agents are performed using a user-centric combination of multimodal devices including traditional GUI software for PC or Web, and modern omnipresent mobile and wearable devices. The communication framework is an extension of the JaCa (Jason/Cartago) Platform for permitting multimodal interaction between BDI agents and users over an SOA microservices architecture. Most work on multiagent software is centred on traditional software architectures and devices like PCs. However, web interfaces and mobile and wearables devices are nearest to users having sufficient computing resources, including CPU, memory sizes, and multimodal capabilities, for permitting a richer human-software agent interaction.

1 INTRODUCTION

In this research paper, we present a novel user-centric multimodal communications framework for BDI multiagent systems applied in a context of registering and scheduling multi-participant events under an automatic and semi-automatic negotiating process.

Currently, there exists many user-centric devices and technologies that could be used for constructing and communicating multiagent systems with users in richer and expressive ways, than used traditionally.

Rich-Content Desktop Applications, Web User Interfaces, Mobile and Wearables Apps offers diverse and rich modalities for a user communicating, like text, images, sounds, text-to-speech, gesture recognition, speech recognition, pose detection with depth cameras among others.

In addition to the advances in multimodal devices, the connectivity technology offers great possibilities for interconnecting systems and applications on heterogeneous devices and platforms. In a way, Service Oriented architecture and the novel Microservices framework offers almost unlimited capabilities for link clients and services.

The principal contribution of our research work is the proposal of JaCa-MM, a hybrid multimodal framework based on a SOA/Microservices architecture for communicating users with agents. This framework permits to building a real-world multimodal application that is user-centric and communicating them with a BDI Multiagent Systems based on standard technologies, facilitating its adoption. The framework is based on Jason BDI Agents and CArtAgo for communicating agents with their environment.

In past several architectures has been proposed (Santi, 2010); (Minotti et al., 2009); (Ricci, 2014), but our approach is user-centric in communication modalities, involved devices and deployment facilities for reach to real-world applications.

For testing our framework, an application for automatic meeting negotiation has been developed

Herrera, J. and Figueroa, H.
JaCa-MM: A User-centric BDI Multiagent Communication Framework Applied for Negotiating and Scheduling Multi-participant Events - A Jason/Cartago Extension Framework for Diary Scheduling Events Permitting a Hybrid Combination of Multimodal Devices based on a Microservices Architecture.
DOI: 10.5220/0006751703180330
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 1*, pages 318-330
ISBN: 978-989-758-275-2

and deployed in several multimodal devices running over the Internet for communicating purposes.

Although the application is of a specific context, the multimodal communication framework could be applied to another multiagent system and even extended for aggregating additional services.

2 BACKGROUND

A multiagent software system is a kind of artificial intelligence application where autonomous, pro-active, reactive "intelligent" software pieces interact between them (social abilities) for reach a common desired objective, by using some reasoning capabilities (Wooldridge et al., 1995).

In literature, several types of agents exist. However, some of the most relevant and used agent types are BDI; this is the framework used in our research work, below we describe this framework and related elements.

2.1 BDI Agents

A BDI agent is a kind of rational or "intelligent" agent represented as a computational entity with certain mental attitudes, Beliefs, Desires, and Intentions, designed for help in the resolution of complex tasks in dynamical environments (Rao and Georgeff, 1995).

Beliefs express the knowledge of the agent from its environment, obtained from sensorial inputs and because of its deliberation processes.

Desires reflect possible environment states that the agent can reach. The desires are treated as options of actions that the agent has.

Intentions represent the desires with the which the agent is committed to performing its actions.

A BDI Agent can be a part of a society of agents, and like in any society, the agents need to communicate between them. This communication process is performed through speech acts (Searle, 1962), a high-level communication expressed as performative verbs to communicate, delegate or interrogate to another agent, among others.

Many agent-oriented programming languages exist, one of the most used, is the combination Jason/CArtAgo, on which is based our framework and application.

2.1.1 Jason

The Jason agent programming language is a Java extension for writing agent-oriented software. Jason is based on AgentSpeak(L), an abstract agent programming language with basis of logic programming, with some extensions for developing practical multiagent systems (Bordini and Hübner, 2005).

In Jason, BDI agents are expressed based on a set of beliefs and plans. Beliefs set are composite from belief atoms with a form of a first-order logic predicate in form p(x) indicating that a subject x accomplish a property p. A typical Jason program starts with an initial belief set, and through the external events and plan execution this belief set are modified, increasing or decreasing.

Intentions in Jason are represented as instantiated plans, where a plan consists of a triggering event (name plan itself), a context represented by a series of beliefs that must be true for permits the plan execution begins; additionally, to trigger event and context, a set of subgoals that the agent must accomplished conforms the body of plan.

Exists two types of goals: achievement goals and test goals. A goal is represented by a predicate with a prefix '!' or '?' respectively. And achievement goal describes a state of the world that is desired by agent. A test goal unifies a predicate with an agent belief, is a type of self-consultation about a determined mental state.

A triggering event indicates when a plan is started, for internal porpoises (a sub goal needs to being achieved) or external for belief updating, resulting from being in contact with the environment. These triggering events can increase beliefs (a predicate with a prefix '+') or decrease beliefs (a predicate with prefix ('-').

Example of beliefs and plans are:

```
work_weekday(monday).
work_weekday(tuesday).
minimum_activity_minutes(15).

+!scheduleMeeting(DOW, D, M, Y,
DUR,H, MIN, PREF) :
        work_weekday(DOW) &
        minimum_activity_minutes(MM)
        & DUR > MM
    <-
   !searchAvailableHours(D, M, Y,
   DUR, LAH);
   .length(LAH, NAH);
   NAH > 0;
   .nth(0, LAH, SH);
   SH = separate(PREF, H, MIN).

-!scheduleMeeting(DOW, D, M, Y,
DUR,H, MIN, PREF) : true <-
.print("Impossible to scheduling a
```

```
meeting in the requested date"); H=-
1; MIN=-1; PREF=-1.
```

In this example three beliefs are defined; the beliefs establish the mental state of the agent to learning that working days are Monday and Tuesday and the minimum length of a meeting is 15 minutes. Additionally, a plan named `scheduleMeeting` is defined, its receives day-of-week (DOW), day (D), month (M) and year (Y) and duration (DUR) for the meeting and instantiates hour (H), minute (MIN) and preferences (PREF). The context for enabling the execution of this plan is that the day of the week was a working day and that the minimum duration of meetings is satisfied. The sub goals of this plan are searchAvailableTime for the day, month and year received and obtained a list of available times according to the availability of the agent, expressed as beliefs. If at least one time is founded, the first is taken and returned to the caller.

2.1.2 CArtAgo

An agent exists in an environment and must interact with it. Jason programming language offers capabilities for modelling and programming agents, however, doesn't offer means for interacting with users and environments for real-world applications. CArtAgo, Common ARTifact infrastructure for Agents Open environment (Ricci et al., 2006), is a framework for programming virtual-environments for multiagent systems that can be integrated to Jason for permitting interacting software agents with users, accessing devices and sharing information or services from another application through the concept of artifacts. An artifact is a piece of software expressed as a Java Class that wraps a set of resources and tools accessible to agents for interacting with its environment (Ricci et al., 2011). Artifacts are grouped in workspaces which could be distributed across a set of network nodes since it's developed in Java; only Java communication is supported. Also, CArtAgo framework offers GUI capabilities for developing applications which communicating with users but limited to Swing Java Applications. Although the CArtAgo communication abilities are limited, and only supported by Java Applications, it is extensible, with the possibilities to increase its capabilities. In Figure 1 we present a traditional architecture including Jason agents and a CArtAgo environment.

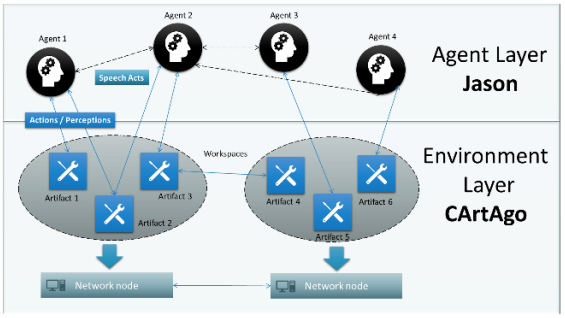

Figure 1: Typical Jason/CArtAgo Framework, adapted from (Ricci et al., 2011).

2.2 SOA and Microservices Architecture

Microservice architecture is a software design pattern where a single application is composite of several small, fine-granularity and loosely coupled "business" services, each one executing in an independent deployed process and communicating between them with lightweight and stateless mechanisms like HTTP and JSON data format (Fowler and Lewis, 2017). Given that microservices are based on web technologies, they are programming language agnostic and omnipresent, facilitating the integration of different platforms services and clients. The commonly used framework for developing microservices is REST (Representational State Transfer), which is a communication mechanism based on HTTP verbs like GET, POST, PUT and DELETE for request operations to read, add, update or delete information to services, using JSON or XML data formats (Richardson, 2017). An example of microservice architecture is presented in Figure 2.

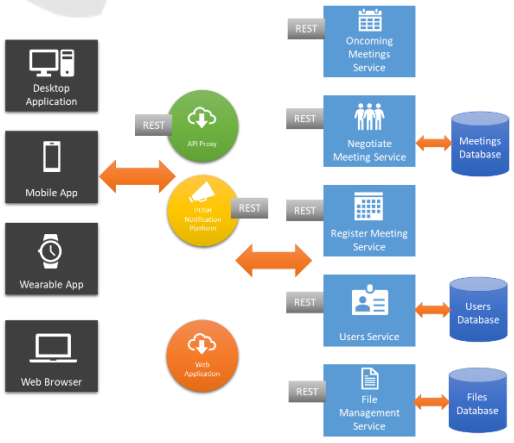

Figure 2: Microservice architecture.

Microservices architecture which is based on Domain Driven Design is a specialization of the Service Oriented Architecture, SOA, design pattern, used for developing Internet-based applications since several years ago. In SOA, an application is composed from a set of related services, having an infrastructure in common and deployed at the same time and normally in the same server infrastructure. In other hand, microservices are loose coupling and independent one from others, even when developed, tested and deployed in different time and infrastructure.

Microservices architecture has permitted to develop applications running in a high variety of client technologies like desktop PC applications, web applications like mobile and wearable apps. This independence of services and clients is possible through the definition of a well-known API over industry-standard protocols like HTTP.

Using a service-based architecture, given that many kinds of devices could access and share information, the architecture empowers and facilitates the use of different input and output modalities of communication by using standard and well-proved mechanisms, improving the user experience.

2.3 User-centric Multimodal Framework

A multimodal interface is characterized by the use of multiple (simultaneously or separately) human sensory modalities supporting combined input/output modes (Sebillo et al., 2009). Multimodal interfaces facilitate multimodal interactions between users and applications, enriching communication process. Most devices have multiple modalities for input and output, in special mobile and wearable devices, and those possibilities could empower the communication of software agent system and users, by permitting a communication style used traditionally between humans through several channels for input and output like voice, gestures, text, images, etc.; like so omnipresence obtained from the presence of devices and telecommunication technologies like Internet.

Traditionally, the development of multimodal applications represents a hard and difficult programming and architecting task due to the complexity of mastering each modality technology augmented by the programming task for developing too many types of computational devices (Dahl, 2013).

For help in this challenging work, the W3C proposes a framework for designing multimodal-applications through the Multimodal Architecture and Interfaces (MMI) specification where a set of components and the typical interaction patterns between them are identified and standardized. Together with standardized communication by passing messages using a standard data format defined in the Extensible Multimodal Annotation (EMMA). In figure 3, we present a schematic view and an example of MMI architecture.

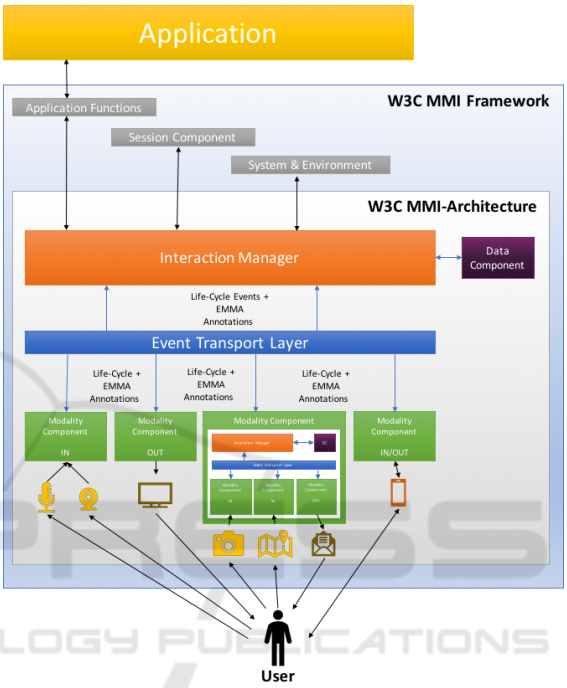

Figure 3: Schematic view of MMI Framework adapted from (Wikipedia contributors, 2017).

2.3.1 Modality Component

Modality Component encapsulates input/output capabilities for different device types, including:

- User input. Keyboard, mouse, gestures, touchscreen, audio, video, photograph, deep sensors, pose detectors, QR codes, and barcodes, etc.
- Sensor input. GPS, temperature, medical sensors, etc.
- Biometric input. Speech verification, fingerprint reader, face recognition, etc.
- Output. Text, graphics, video, audio, paper print, vibration, text-to-speech, etc.

Each component can be associated with a single device for input or output or could be even more complex through the association with several devices or modalities for integrating them and provide a unique stream of data, for example, synchronize voice

inputs with face recognition for user intents recognition.

Even more, a modality component could incorporate in a recursive form, a complete multimodal processing for example speech recognition plus a natural language processing offering a text output from the recognized audio speech.

2.3.2 Interaction Manager

The interaction manager is responsible for coordinating the communication between the application and the user. It has the only component that can interact with modality components through life-cycle events with EMMA annotations. For helping in its job, a Data Component could exist providing information specifically for the integration process.

2.3.3 Data Component

The Data Component is an optional part of the architecture, if it exists, then it has the mission to provide all the necessary information to the Interaction Manager for doing its job. Examples of that information could be user identification, modalities preferences, user context (location, language, sex, title, etc.), among others.

2.3.4 Event Transport Layer

The Event Transport Layer is the responsible for managing message events between the Modality Components and the Interaction Manager. Normally, the messages exist in a request/response pairs.

The communication is based on Life-Cycle Events originated from the Interaction Manager or a Modality Component. Two kinds of events exist, generic controls or modality specifics. A generic control-event is like start, pause, resume or stop transfer request, and the response is the acknowledge of the request. A modality-specific event is used to set the configuration of a transfer and the response obtained from a determined modality like voice, video, text, location, etc.

The semantics of the communication process is maintained using EMMA annotations. An example of an event response is the following:

```
<mmi:mmi
xmlns:mmi=http://www.w3.org/2008/04/mmi-arch version="1.0"
xmlns:emma="http://www.w3.org/2003/04/emma">
  <mmi:doneNotification
mmi:source="userResponseToConfirmation"
   mmi:target="requestResponseFromUser"
mmi:context="meetingNegotiation"
   mmi:status="success"
mmi:requestID="123456" >
   <mmi:data>
   <emma:emma version="1.0">
    <emma:interpretation
    id="int1"
    emma:confidence=".80"
    emma:medium="acoustic"
    emma:mode="voice"
    emma:function="confirmation">
    <response>accepted</response>
    <meeting>
      <date>10/10/2017</date>
      <start>10:30</start>
      <duration>45 minutes</duration>
    </meeting>
    </emma:interpretation>
   </emma:emma>
   </mmi:data>
  </mmi:doneNotification>
</mmi:mmi>
```

2.3.5 Application and Runtime Framework

The application is the final target and origin of the multimodal interaction; it could be a traditional desktop software, a web application, a mobile/wearable app or a multiagent software.

An application exists in an environment and requires a set of infrastructure services like communications, session management, user authentication, among others. These components are announced in the framework but not defined, permitting adapting it to any infrastructure or application needs. For example, a CArtAgo artifact could communicate with an Interaction Manager for multimodal interaction with agents over a microservices infrastructure for communicating with several device and application types.

2.4 Related Work

This research work, combines multimodal interaction with a Jason/CArtAgo multiagent system over a SOA/microservices design framework. Several works tackle some of these aspects in the past, however, we couldn't find work with exactly same objective and for this reason, we say, this is a novelty approach. We present an architecture and an application sample that uses this proposed

architecture. Some of the related works used even as a basis for our framework and are mentioned below.

Several CArtAgo extensions have been proposed in the past for combining multiagent systems to mobile platforms. Examples of these proposals are like the port to Android Operating System described in (Santi, 2010), and for running over platforms like Web (Minotti et al., 2009) or working with SOAP Web Services (Ricci, 2014), which was the original form of SOA Applications, unfortunately, the support of these projects were stopped years ago, we think that a modern microservices architecture letting to a multiagent system reach different devices and platforms without port the platform itself.

In the subject of multimodal multiagent system architectures, approaches like (Dulva et al., 2011), (Griol et al., 2013) and (Sokolova et al., 2015) has been presented for specific multiagent systems.

3 MULTIMODAL MULTIAGENT COMMUNICATION FRAMEWORK

In this section, we describe our multimodal multiagent framework based on a microservices architecture. Our approach is an extension of the Jason/CArtAgo programming platform. In Figure 4 we present it graphically.

The framework is composed of four layers: Intelligent Multiagent System Layer, Agent Environment Layer, SOA/Microservices Layer and Multimodal Device Layer, which are described below.

3.1 Multimodal Device Layer

The human-computer interaction is performed in the multimodal device layer, processing every input from any of the supported modalities, interpreting data, encapsulating both, the event and interpreted data into an EMMA annotation and delivering it through the SOA/Microservices Layer to the respective agent. Given that a Jason agent doesn't understand EMMA, a CArtAgo artifact, in the Agent Environment Layer, translate the multimodal message to an and/or belief and sends to an agent.

When the agent needs communicate or questioning something to the user, it throws an event to its environment artifact. This artifact delivers to the network the request using a PUSH notification service for reach the respective user device, the message is interpreted, and the convenient output modalities are selected and used for communicating with the user. More details on PUSH notifications in section 3.2.

Table 1: Input / Output Interactions between a user and the multiagent system.

Action	Description	Type
Querying Meetings	Perform a query of the registered meetings of the user to the agent associated with the user. (GET verb)	Input
Registering Meeting	Request the register of a new meeting with the appropriate data (possible dates, duration, users invited, attached files). (POST Verb)	Input
Updating Meeting	Modify data associated with a registered meeting. (PUT Verb)	Input
Deleting Meeting	Request the removing of a meeting registered by the user. (DELETE Verb)	Input
Notification Action Result	The agent communicates to the user the result of a requested action. (POST Verb)	Output
Request Confirmation	The agent request to the user the confirmation of a meeting. The confirmation is necessary when the automatic negotiation between agents fails to find a time, and manual registration is required. (GET Verb)	Output

Table 2: Devices and supported modalities.

User Interface	Modality	Type
Desktop Application & Web Application Computer	Typing and pointing Voice	Input
	Text & Image Sounds Text to Speech	Output
Mobile device	Finger touch and gestures Voice	Input
	Text & Image Sounds Vibration Text to Speech	Output
Smart watch (Wearable)	Finger touch and gestures Voice	Input
	Text & images Vibration	Output

Any multimodal device HTTP-speak-capable could be a client of the platform. This generalization is possible by using a SOA/Microservices architecture. Almost every device has the possibility of connecting to the Internet and the web, and only that is required for using in our framework. Internet of the Things (IoT) advanced so much in this way, permitting integrating a high variety of technologies.

In Table 1, we are presenting all the actions that a user could perform in the application and the possible outputs requirements by the agents for which a user must respond.

The Table 2 summarizes the user interfaces/devices and modalities supported by our framework.

3.1.1 Input Processing

For every user interface and device supported in our framework, we rely on native multimodal SDK, in Table 3 we summarized the used technologies and used APIs.

Table 3: Input modalities and technologies applied.

User Interface	Modality	Type
Desktop Application	Typing and pointing	Windows Mouse and Keyboard Support (.NET API)
.NET WPF Application	Voice	Microsoft Speech API
Web Application	Typing and pointing	HTML/CSS/JavaScript
	Voice	HTML 5 Speech Recognition API
Mobile device	Finger touch and gestures	Android SDK/ iOS Cocoa SDK
	Voice	Android/iOS TTS
Smart watch (Wearable)	Finger touch and gestures	Android Wear SDK
	Voice	Android Wear SDK

The process starts when an event input has been received from the user, in one of the supported modalities. The raw input data is processed into the corresponding Modality Component, and when is complete, it sends through the Event Transport Layer to the Interaction Manager and the information obtained is encapsulated into an EMMA annotation. In each Modality Component, processing is performed depending on the input and the device, for example, using the native or the appropriate libraries, a speech recognition process could be performed, but only the recognized text is the output of such Modality Component and sent to the following components.

When received for the Interaction Manager, the event and the associated data are analysed, and if they are significant to the application, then are encoded used the EMMA Proxy and sent to the appropriate Microservice using the respective client module.

The client module packs the EMMA annotation into a RESTful message in JSON format and sending to the server using the adequate HTTP verb (GET, POST, PUT or DELETE). The selection of the verb depends on the required actions, for example, a GET verb is used when a request of information is required, a POST request for register new meetings or file attachment, a PUT for updating data and DELETE for remove a previous meeting.

In all platforms, a Data Component helps in the interpretation, encoding and correct routing of the event. This data component, store facts like user identity, device state, registered meetings, information for locating servers and user preferences of communication modalities.

3.1.2 Output Processing

When an agent has something to saying or to questioning to a user, then a PUSH notification is sent and received in the corresponding device for its processing. When a PUSH notification is received for in the client application, a GET request is made to the respective service for recover the complete output information; each notification is sent with two values, the type of event and a communication ID. This ID is used to obtain the data from a Service about the communication event needs.

The information of the event is encoded into an EMMA annotation by the proxy and is sent to the Interaction Manager for selecting the appropriate output modalities.

With the help of the Data Component, the Interaction Manager determines the preferences of the user, like language, gender, preferred sounds, etc., and prepares the communication data. This data is sent to the corresponding Modality Component for showing it to the user and if it's necessary, ask him a question according to the necessities of the agent, then, receive the input from the user and send back to the agent using the process described in the previous subsection. Like the input processing, the output is performed by the native capabilities of the platform and device. Use of text, images, sounds or any other resource depends on the device. Some complex output process like Text-To-Speech is used when available in the corresponding device.

3.2 SOA/Microservices Layer

The SOA/Microservices Layer presents a middleware component in the overall framework; its function is connecting the multiagent system with all the multimodal user interfaces and devices supported by the application.

Three different components belong to this layer: Services Proxy, Microservices, and a PUSH notifications service.

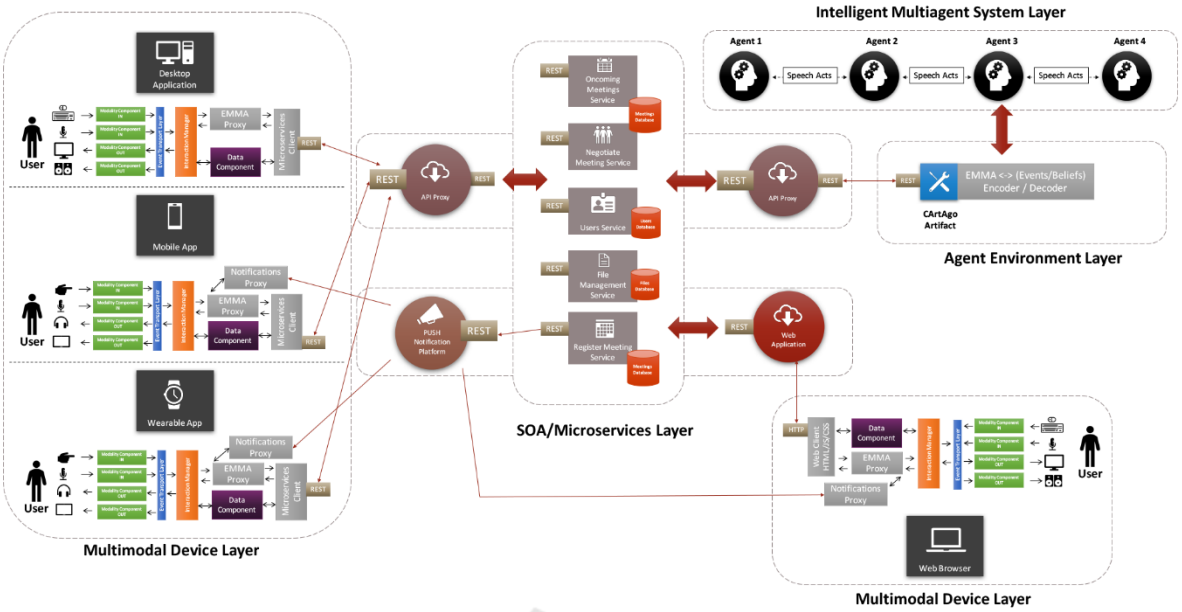

Figure 4: JaCa-MM Framework.

3.2.1 Services Proxy

The services proxy is a routing component used to connect devices with microservices. This component receives a request sent by a user interface/device or by an artifact belonging to the Agent Environment Layer.

Its work is deciding which action is required, receives a set of parameters according to the action and routing it to the respective service. This layer encodes and decodes data in JSON format, calling the services using a RESTful invocation and passing the data for the adequate processing.

When a REST architecture is used, and API is defined, the API implementation is a set of

Microservices and this component are in charge to call that API, for this, is considered a client. All communications are made using HTTP protocol with the appropriate verbs like defined in Table 1.

3.2.2 Microservices

The Microservices are the heart of the middleware layer, its represent the actions that a user can request form an agent, and the indications that an agent can communicate to and user through a device and its UI.

The microservices layer is language agnostic, in this implementation Java EE 7 running on a glassfish server was used, but practically any modern Web Platform could be used for implementing them. This benefit of being independent of language and platform is due to be based on standards. HTTP is a standard protocol, REST is based on it and gives a semantic context for each one of the verbs supported by HTTP. In another hand, JSON is the most used data representation language used nowadays, is clear and easy to implement and interpreting and most languages and platforms support it.

Five services were defined for this application: Oncoming Meeting, Negotiate Meeting, Register Meeting, Files, and Users.

The Oncoming Meeting Services is responsible for querying the Registered Meetings Database. This service is invocated both the users and the agents. The user for query and take decisions and establish remainders for the oncoming meetings. The agents use this information to aggregate these registered events as beliefs in its mental state at the starting point, and those beliefs are used principally for the process of automatic negotiation between agents. The database of meetings incorporates a Meeting ID, a Name, a Start Date, a Start Time, an End Date, an End Time and the Invited Users.

The Negotiate Meeting Service is invocated from a user from the multimodal device for requesting to his agent to start the negotiation of a meeting in a range of dates, with a subject and a list of requested users. The negotiation process only involves the agents of the users requested for the meeting. This service communicates with a CArtAgo Environment Artifact to send the appropriate event to the agent and throws the negotiating process between agents.

When a meeting has been agreed between agents or in the worst case, by the users themselves, the Register Meeting Service store in the database the new agreed meeting and send PUSH notifications for confirming the new event to the participants. This Service is invocated for the agent through the Environment Artifact when the negotiation has been concluded.

A meeting request can include files like word processing documents, spreadsheets or presentation files, the File Service is the responsible for storing and recovering that files when a new meeting is registered or when some of the previous registered are modified. Through the user interface a user can request the attached documents, and they are sent to this service.

The User Service permits authenticate a user, store preferences of each one and maintains information for linking users, devices and agents through the framework.

3.2.3 PUSH Notification Service

In this framework, we assume that agents, its environment artifacts, and microservices reside on well-known servers. And given this assumption is easy for user devices locating the microservices servers and these find and communicate with agents and artifacts.

However, when an agent needs to communicate with the user devices, specifically with non-always connected devices, like Web applications, Mobiles, and Wearables, only the Microservices Layer is not sufficient. For this reason, another component is necessary, the PUSH Notification Service.

A PUSH Notification service is a kind of server software that can feed notifications to devices and web applications, even when not always connected. For doing its job, when an App is installed on the device, this is registered in the PUSH platform, and an ID Token is assigned for it when is necessary to contact the device, the platform could do it through a resident mini-server installed on the device and the appropriate ID Token.

This mini-client is installed as a service when the App is installed on mobile and wearable devices and is requested to the user to subscribe it when he enters for the first time to a web application.

These notifications are not necessary in a context of a Desktop Platform, given an Application could always be connected to the server.

This layer is the most commonly used in commercial applications on mobile, wearable and web applications nowadays.

3.3 Agent Environment Layer

The Agent Environment Layer is a set of Java Classes compatible with CArtAgo framework specification. A BDI Agent like Jason Agents works with mental attitudes like Beliefs, Desires and Intentions, Beliefs and Plans in Jason. In other hand user interfaces on devices maintains and understand multimodal communication events and EMMA data which are routed to Microservices for reach to the Agents.

The principal job of this layer is to create a virtual environment for the agents and through it, communicating with real-world environments.

It must receive events from agents and translate them to requests for the appropriate microservice for routing event and data to the user.

In another hand when a user requires for an action from the agents, or respond to a previous information questioning, the EMMA annotated data must be translated to events and register or remove beliefs in the mental state of the agents.

For doing its translate job an EMMA - (Event/Beliefs) Encoder/Decoder is incorporated into the artifact. For actions required from a user, the translation is an event for trigger plan execution on Agents. When data is proportionated from the user, a Beliefs Update is necessary, for add or remove beliefs of an agent. When the action is required by the agent, event and beliefs are encoded into an EMMA annotation and sent to the device through a PUSH notification.

Given the middleware nature of these components, they must understand two different domains, the agent domain based on beliefs, plans and goals, and the microservice domain, based on API utilization through REST requests.

3.4 Intelligent Multiagent System Layer

This layer is the one which encapsulates the “Intelligent” behaviour of the framework. It represents the Multiagent System, the reason for design and construction of the overall architecture.

An Intelligent Multiagent System in our framework is compositing of BDI Agents; this layer is composed of Jason Agents that communicate between them using Speech Acts. Each Agent has a mental state, based on attitudes like Beliefs, Desires, and Intentions.

In this framework, each user has assigned an Agent who is listening for requests of actions like:

- Sending the oncoming meetings of a certain period.

- Negotiate a new meeting between a group of other agents associated with the users requested to the meeting.
- Registering the agreed meeting in the accorded date and time.
- Update the information of a registered meeting and communicating to another agent for notifying to their respective users.
- Delete a registered meeting and communicating to another agent for notifying to their respective users.

An Agent starts loading the user-assigned preferences (sex, title, name, start working time, end working time, in-week working days, resting days) and the oncoming meetings of the current date and the next 15 days; this information is registered as the original beliefs of the agent.

In addition to beliefs, each agent has a set of plans which describe how to reach to the state of the world desired, described as a set of goals to execute. These plans are executed from a request of the user, received by the corresponding artifact in the Environment Layer or by a requesting from another agent.

The communication between agents is performed by sending and processing speech acts. In Table 4 we summarize the speech acts and associated plan used for communicating the agents between each other and with the Environment Layer.

In Figure 5 we present the negotiating and register meetings actions in a UML Sequence Diagram with the respective plans and speech acts.

Table 4: Speech acts used in agent communication.

Speech act / Plan	Description
negotiateMeeting	(Plan) Start the process of negotiating a meeting with all the requested participants. First obtains the available times in a period from the other agents, and proceed to deliberate for finding a common time for meeting
requestAvailableTime	(Speech Act) The negotiating agent request from the others agents the list of available time sufficient for the requested meeting
findCommonTime	(Plan) Process the available times from participants in a specific date for finding one in common
scheduleMeeting	(Speech Act) Register a meeting and communicate it to another agent for doing the same
commonTimeToAll	(Plan) A deliberation process starts to determine if a selected time is common to all participants and proceed to registration
loadOnComingMeetings	(Plan) Loads the oncoming meetings from a database through an artifact
searchAvailableTime	(Plan) Performs a search for a common time in the list of available times from all participants, if one is founded, it is corroborated by the participants for confirming it. If no common time founded, an interrogation process starts with the users from candidate times and if all agreed, the meeting is confirmed.

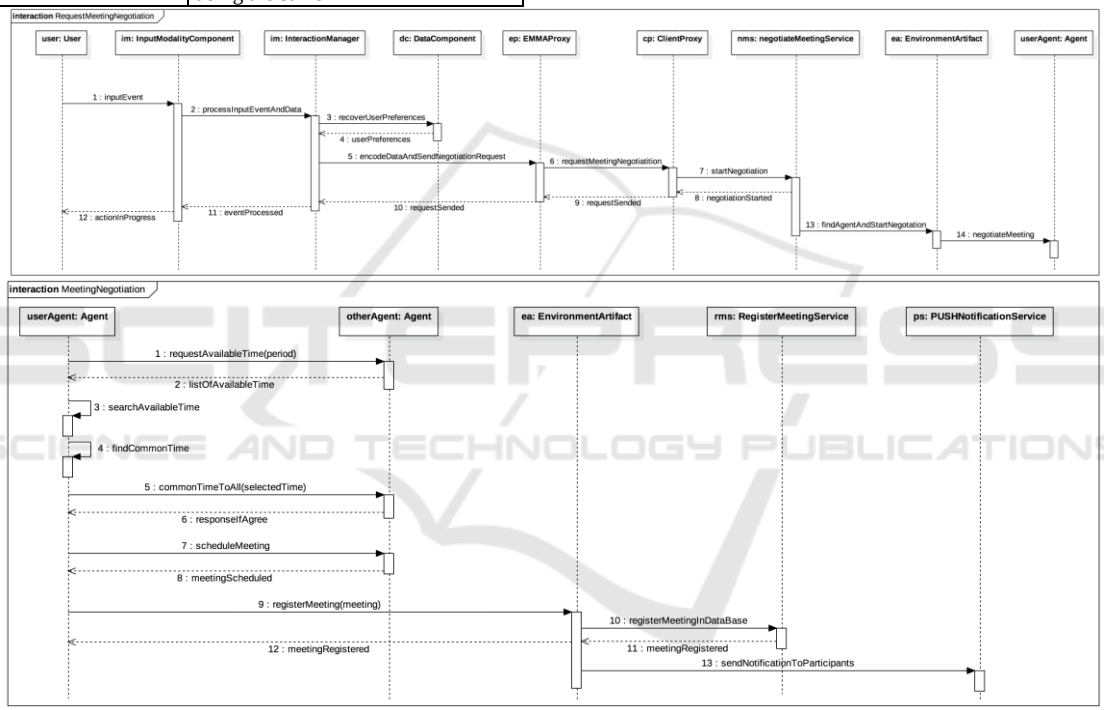

Figure 5: Sequence Diagram of Meeting Negotiation Process.

4 RESULTS

The proposed framework was implemented and tested in twenty-four meeting negotiating process, previously the user preferences and some artificial meetings were registering to try the negotiating process with success and failures. Eight users tested the application in the four platforms mentioned in section 3.1. Each user submits three meetings in a random way, deciding date, duration and participants of each event. When the process of submitting meeting began, all request could be negotiated automatically by the agents, conform more meetings were registered, the automatic procedure failed, and the questioning-users process was necessary to operate. In Figure 6 we present the statistics of the test of negotiating-process.

With respect to the input modalities, as it was predictable the use of mouse and keyboard on the Desktop and Web Application was the most precise of all the implemented modalities. In all platforms speech recognition was enabled, obtaining a high-performance recognition on PC with a Desktop Application, in second place was the Web Interface, a lower precision in mobile device (smartphone and tablet) and the worst was in the wearable (smart watch).

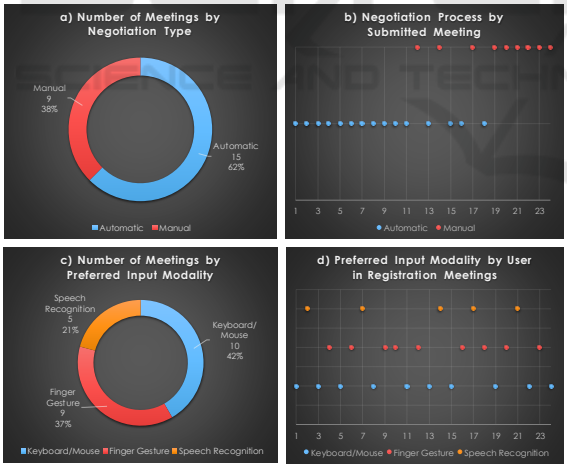

Figure 6: Statistics obtained from the testing process of the application. a) The number and percent of meetings according the negotiation type; b) Shows the negotiation type, blue for automatic and red for manual for each meeting; c) The number and percent of meetings according to the preferred modality by each user; d) Show the preferred modality for each meeting.

However, the users expressed that they could submit meetings or respond to agent question better on mobile and wearable devices with Internet connection, although the input precision was not optimal. Although voice recognition fails many times, the finger gesture recognition works very well. In their opinions, a well-designed user interface using the multimodal capabilities was adequate for doing this task.

5 CONCLUSIONS

According to our implementation experience and the test with real users and their opinions, we could affirm that the use of multimodal user interfaces over different devices enriches the user-agent communication and for that reason, the utility of a multiagent system. Implementing a system over several platforms is a challenging task. However, a well-designed architecture combining an "intelligent" multiagent system, with the existing resources in user-experience design, device-capabilities and modern communication technologies like SOA/Microservices Architecture, offers great benefits for researchers by delivering software to solve real-world problems to real-world users. These benefits are mutual; the users obtain "Intelligent" Applications to solve their tasks, and by another hand, then researchers obtaining benefits for testing "intelligent" algorithms on real-world scenarios.

Mobile and wearable devices still must improve on certain multimodal recognition capabilities, however, the possibilities to be everywhere give them great benefits and opportunities to deploy multiagent input and output extensions.

The SOA/Microservices architecture was the glue between "non-intelligent" multimodal applications and the layer of "intelligent" multiagent system. This architecture permits the deployment of solutions over networks like the Internet and reaching practically any user in any modern device.

REFERENCES

Rao, A. S., Georgeff M. P., 1995, BDI Agents: From Theory to Practice, *Proceedings of the 1st International Conference on Multiagent Systems*. AAAI.

Searle J. R., 1962. Meaning and speech acts, *The philosophical Review*, 71(4): 423-432.

Bordini R.H., Hübner J.F., 2005, BDI Agent Programming in Agent Speak Using Jason. In: Toni F., Torroni P. (eds) *Computational Logic in Multi-Agent Systems*. CLIMA 2005. Lecture Notes in Computer Science, vol 3900. Springer, Berlin, Heidelberg.

Wooldridge M., Jennings N. R., et al., (1995). Intelligent

agents: Theory and practice. *Knowledge engineering review*, 10(2):115–152.

Ricci A., Viroli M., and Omicini A., 2006, Construenda est cartago: Toward an infrastructure for artifacts in MAS. *Cybernetics and systems*, 2:569–574.

Ricci A., Piunti M. and Viroli M., 2011, Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192.

Fowler M. and Lewis J., 2017, Microservices a definition of this new architectural term. *URL: http://martinfowler.com/articles/microservices.html*, Last accessed: Feb 2017.

Richardson, C., 2017, Microservice architecture pattern. *microservices.io*. Retrieved 2017-03-19.

Sebillo M., Vitiello G. and De Marsico M., 2009, Multimodal Interfaces, *Encyclopedia of Database Systems*, pp 1838-1843, doi 10.1007/978-0-387-39940-9_880.

Dahl D. A., 2013, The W3C Multimodal Architecture and Interfaces Standard. J. on *Multimodal Interfaces, Volume 7, Issue 3*, November 2013 (published online April 13, 2013. ISSN: 1783-7677 (Print) 1783-8738 (Online).

Wikipedia contributors, 2017, Multimodal Architecture and Interfaces, *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Multimodal_Architecture_and_Interfaces&oldid=800595600 (accessed November 6, 2017).

Santi A., 2010. JaCa-Android, *http://jaca-android.sourceforge.net/*.

Minotti M., Piancastelli G., Ricci A., 2009. An Agent-Based Programming Model for Developing Client-Side Concurrent Web 2.0 Applications, *5th International Conference on Web Information Systems and Technologies (WEBIST 2009)*, 23-26 March 2009.

Ricci A., 2014. cartago-ws, *https://sourceforge.net/projects/cartagows/*.

Dulva H. M., Tadj C., Ramdane-Cherif A., and Levy N., 2011. A Multi-Agent based Multimodal System Adaptive to the User's Interaction Context, Multi-Agent Systems - Modeling, Interactions, *Simulations and Case Studies, Dr. Faisal Alkhateeb (Ed.), InTech*, DOI: 10.5772/14692. Available from: https://www.intechopen.com/books/multi-agent-systems-modeling-interactions-simulations-and-case-studies/a-multi-agent-based-multimodal-system-adaptive-to-the-user-s-interaction-context.

Griol D., García J., Molina J. M., 2013, A multi-agent architecture to combine heterogeneous inputs in multimodal interaction systems, *Conferencia de la Asociación Española para la Inteligencia Artificial*, Multiconferencia CAEPIA 2013: 17-20 sep 2013. Madrid: Agentes y Sistemas Multi-Agente: de la Teoría a la Práctica (ASMas). (pp. 1513-1522), ISBN: 978-84-695-8348-7.

Sokolova M.V., Fernández-Caballero A., López M.T., Martínez-Rodrigo A., Zangróniz R., Pastor J.M., 2015, A Distributed Architecture for Multimodal Emotion Identification. In: *Bajo J. et al. (eds) Trends in Practical Applications of Agents, Multi-Agent Systems and Sustainability*. *Advances in Intelligent Systems and Computing*, vol 372. Springer, Cham.