

Standardization of communication among smart machines and its implementation with FIPA compliant agent-based systems

A dissertation presented by
Saurabh C Mandaokar

Supervised by
Prof. Makarand Kulkarni



Submitted to
Indian Institute of Technology, Bombay

for the degree of
Master of Technology
In
Manufacturing Engineering

June 2021

Declaration

"I declare that this written submission represents my ideas in my own words, and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated, or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action as per the rules of regulations of the Institute."



Saurabh Mandaokar

193100081

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Prof. Makarand S Kulkarni

(Signature of Thesis Supervisor with date)

Acknowledgement

I would like to express my sincere gratitude towards Prof. Makarand S Kulkarni for believing in me to carry out this project work under his supervision. His constant encouragement, friendly interactions, constructive support have enabled this work to achieve its present form. His innovative perspective towards things and his continuous pursuit of perfection has had a profound effect on me and has transformed me majorly.

I would also like to thank my colligue, Mr. Aniket Adsule, for his help throughout the course of my work towards achieving the project deliverables. A special thanks to Prof. Asim Tewari for helping me to come up with a long term vision for the project and inviting me to continue my work at NCAIR lab of IIT Bombay.

Saurabh Mandaokar

Abstract

Industry 4.0 encourages the use of autonomous machines with decentralized decision-making capabilities.

These machines collaborate to self-organize and distribute the allotment as well as execution of tasks. Therefore, there is a need for some form of communication mechanism that not only facilitates data sharing among network entities but also enables them to interact in a common language. This language must have a well-defined vocabulary and semantics, well known to each entity in the network. The interactions between autonomous smart machines in an industry are important to enable them to take decentralized decisions to achieve a higher degree of interoperability.

This project work aims at introducing an open standard for interaction between autonomous machines. The language of interactions is based on a standard known as FIPA-ACL. FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. FIPA – Agent communication language (FIPA-ACL) has clearly defined interaction patterns called “interaction protocols” and a set of communication acts known as “Performatives” that are used by agents to have a dialogue.

The demonstration of these interactions is done through Agent-based system architecture. Each Agent is a node in a network that represents a single smart entity in an industrial environment or enterprise. These agents resemble in their basic structure, which includes connecting to the main server on initiation and running its predefined behaviors. Each agent can be further custom programmed according to the specifications of the system it will be serving for.

Keywords: Industry 4.0, smart manufacturing, Cyber-physical systems, communication, interoperability, FIPA-ACL, agent-based systems, python agent development platform, web-sockets based MAS, multi-agent system in python.

Table of contents

Chapter 1: Introduction	08
1.1 Smart manufacturing	08
1.2 Smart machines	08
1.3 Properties of a smart machine	08
1.4 Using a standard that defines a fixed set of properties for a smart machine	10
1.4.1 Identification	11
1.4.2 Industry 4.0 semantics	12
1.4.3 Virtual description	13
1.4.4 Industry 4.0 services and states	14
1.4.5 Standard functions	14
1.4.6 Security	15
1.4.7 Industry 4.0 communication	15
1.5 Current scenario of communication between devices in industries	16
1.6 Some benefits of interactive machines	16
1.7 Requirement of standards for communication	20
 Chapter 2: Literature review	 21
2.1 What is industry 4.0	22
2.2 Interoperability in context of Industry 4.0	22
2.3 Challenges in Implementation of Interoperability	22
2.4 How standardization in industry 4.0 can help enhancing interoperability	23
2.5 Need of multi-agent systems (MAS) approach	23
 Chapter 3: how to enable machines to interact	 23
3.1 Basics of device interactions	23
3.2 Messaging Format	23
3.3 Message transport mechanism	24
3.4 Messaging Language	24
3.5 Why multi-agent systems	24
 Chapter 4: messaging language	 26
4.1 Knowledge Query and manipulation language (KQML).....	26
4.2 FIPA-ACL.....	27
4.3 Concept of a performative	28
4.4 Description of performatives	28
4.5 Concept of Interaction protocol	29
4.6 Various interaction protocols	29
4.6.1 Request Interaction Protocol	29
4.6.2 FIPA Contract Net Interaction Protocol	30
4.6.3 FIPA Subscribe Interaction Protocol	31
4.6.4 FIPA Request-When Interaction Protocol	32
4.6.5 FIPA Query Interaction Protocol	33
4.6.6 FIPA Propose Interaction Protocol	33
4.6.7 FIPA cancel meta protocol	34
4.6.8 FIPA recruiting interaction protocol and FIPA broking interaction protocol	35

4.7 Concept of performative types	36
4.8 FIPA-ACL message structure	36
4.8.1 Decryption of parameters	37
4.9 Case example for how FIPA interactions take place	38
 Chapter 5: A FIPA compliant agent-based system architecture	 40
5.1 The proposed multi-agent system Architecture	40
5.2 Web-sockets based client to server communication	40
5.3 Agent description File	41
5.4 Online and network services.....	41
 Chapter 6: Server of multi-agent system	 43
6.1 Server network access	43
6.2 server initiation, handling new initiated agents	43
6.3 How server handles messages from the Agents	44
 Chapter 7: Agent	 45
7.1 Machine Agent architecture	45
7.2 Role Agent and its architecture	46
7.3 Agent functions	49
7.3.1 Base Functions	49
7.3.2 Downloadable Functions	49
7.3.2 a) Active Functions	50
7.3.2 b) Passive Functions	50
7.4 Agent startup or agent initiation	50
7.5 Transfer of messages between agents	51
7.6 Function classification	51
7.7 Major base libraries or base functions of an Agent	52
7.8 Message handling	52
7.8.1 Parent message handler	53
7.8.2 Protocol handler functions	53
7.9 Trigger identifications	54
7.10 Identification of target agents	54
7.11 Conversation starter function	55
7.12 Conversation logging	55
 Chapter 8: Integration of agent functions into the agent	 57
8.1 how agents can be customized using downloadable functions	57
8.2 Agent software files storage System	57
8.3 generation of mapping and pointing files	58
8.4 Function Management Layer	59
 Chapter 9: system robustness	 62
9.1 Hot standby server	62
9.2 sudden machine agent stoppage	62
9.3 standby role agent	62

Chapter 10: Previous approaches used and their drawbacks...	64
10.1 PADE.....	64
10.2 Drawbacks and challenges while implementation of PADE.....	64
10.3 SPADE.....	65
10.4 Drawbacks and challenges while implementation of SPADE.....	66
Chapter 11: Conclusion	67
11.1 What has been achieved on the server-side	67
11.2 What has been achieved at the Agent side	67
Chapter 12: Future scope	69
12.1 On the server-side	69
12.2 On the agent side	69
12.3 Installable application	69
12.4 Agent management	69
12.5 Installing new function package to the Agent-based system (decentralized)	70
References	71

List of Tables

Table 1.0: properties of smart machines have a mention in various literature
Table 1.1: Requirements for the property of Identification for a smart product
Table 1.2: requirements of industry 4.0 semantics for a smart product
Table 1.3: requirements of Virtual description for a smart product
Table 1.4: requirements of Industry 4.0 services and states for a smart product
Table 1.5: requirements of standard functions for a smart product
Table 1.6: requirements of security for a smart product
Table 1.7: Communication between machines and Cloud services/main system software/data repository
Table 1.8: Communication between machine to another Machine
Table 1.9: Communication between machine and operator
Table 1.10: Communication between machine and MHS
Table 1.11: Communication between machine and inventory storage system
Table 1.12: Communication between machine and job
Table 4.1: all FIPA-ACL performatives
Table 4.2: examples of performative types
Table 4.3: FIPA message structure

List of figures

- Figure 1.1: deriving criteria for industry 4.0 product from RAMI 4.0 architecture
- Figure 3.1: basic types of the network model
- Fig 3.2: representation of JSON format
- Fig 4.1: Interaction pattern of FIPA request protocol
- Fig 4.2: FIPA contract-net interaction protocol flow
- Fig 4.3: FIPA Subscribe protocol flow
- Fig 4.4: FIPA request-when interaction protocol flow
- Fig 4.5: FIPA query protocol flow
- Fig 4.6: FIPA Propose Protocol flow
- Fig 4.7: FIPA Cancel-Meta Protocol flow
- Fig 4.8: Proxy interaction
- Fig 4.9: Outer envelope syntax of the FIPA-ACL message
- Fig 4.10: example of interaction for contract-net protocol
- Fig 5.1: Proposed Multiagent Architecture
- Fig 5.2: an Agent file
- Fig 6.1: server initiation and handling new initiated agents
- Fig 7.1: architecture for a machine agent
- Fig 7.2: architecture for a role agent
- Fig 7.3 GUI for job manager agent
- Fig 7.4: abbreviations of functions and their hierarchy
- Fig 7.5: Tasks at agent initiation at the individual system level
- Fig 7.6: serialization and deserialization of a message
- Fig 7.7: All message parsing functions
- Fig 7.8: deciding target agents by target agent identifier
- Fig 7.9: Conversation starter function and other base functions that it uses
- Fig 7.10: conversation log for a completed conversation that took place with request interaction protocol
- Fig 7.11: the flow of function installation procedure
- Fig 8.1: Structure of a downloadable function
- Fig 8.2: Agent folder file system
- Fig 8.3: Mapping and pointing operation
- Fig 8.4: description of introduction function (internal) of a downloadable function
- Fig 8.5: the process of selecting an executable function
- Fig 8.6: type to function mapping file
- Fig 8.7: function type to function pointing file
- Fig 8.8: inter function communication
- Fig 8.9: robustness against role agent failure

Chapter 1: Introduction

1.1 Smart manufacturing

smart manufacturing is the next generation manufacturing paradigm, that makes use of cloud computing infrastructures, smart sensors, artificial intelligence, advanced robotics, big data analytics and additive manufacturing to achieve cost efficiency, flexibility, ease of compliance, better resource management and to improve manufacturing productivity.

According to the National Institute of Standards and Technology (NIST) “Smart Manufacturing are systems that are fully-integrated, collaborative manufacturing systems that respond in real time to meet changing demands and conditions in the factory, in the supply network, and in customer needs.”[1]

1.2 Smart machines

The term smart machine, has no generalized definition. The term itself has its manifestation with different names in various literatures. The common terminologies such as “Cyber Physical Systems” (CPS) and “industry 4.0 products” have similar description to that of smart machines.

A Cyber- physical system consists of both computational and physical components working together to implement a process in real time. CPS creates a collaborative infrastructure to support the digital representation of information along the product and process life cycle. These systems connect sensors, actuators and systems through a high-fidelity network, which adds system functionalities, such as real-time data transfer and status monitoring, allowing for interaction with other systems.

In the literatures, the interpretation of smart machines by the authors is stated by the properties or characteristics of smart machine that they consider. The subsequent section mentions the desired properties in these smart machines.

1.3 Properties of a smart machine

The following properties of smart machines have a mention in various literatures. These also include the properties for “CPS”, “smart devices” and “industry 4.0 component”.

SR. NO	PROPERTIES	REFERENCES
		A:[2] B:[3] C:[4] D:[5] E:[6] F:[7]
1	Human in loop	B, C, D
2	Predictive analytics/predictability/predictive intelligent system	D, E, F
3	Integrating heterogenous systems/ connectivity	C, D, F
4	Adaptive reconfiguration/ adaptability	C, F
5	dependability	C
6	Interoperability/plug and work	C, E, F

7	Interface with preexisting systems/reusable design	C, E
8	Cyber security/data security/privacy	A, C, D, E, F
9	Dynamic nature/resilience	C, D, E
10	Concurrency/concurrent computation	C, D
11	scalability	C
12	Decentralization/decentralized data assessment/ decentralized operation	B, E, F
13	Real time information system	D
14	Integrating physical and computational system	D
15	Self-assessment	E
16	Modularity	E, F
17	Digital mobility	E
18	Autonomous decision making	F, A
19	Identity/ identification	B, F
20	compositionality	F
22	Status	F, A
24	I4.0 compliant services	A
25	Virtual description	A, B
26	semantics	A

Table 1.0: properties of smart machines have a mention in various literatures

Following are the descriptions of such properties identified in the above table.

1.Virtual description: the “digital” version of the production system/module has to be filled with content (e.g. behavioral models, simulation capabilities, predictive conditions)

2.Identity: Smart machine should be uniquely identifiable that signifies its own digital presence, thus providing with a unique identification in the digital world, for example, a network interface address.

3.Modularity: The system is distributed as a set of distinct modules that can be developed independently, and then plugged together. Each of these modules may also represent individual autonomous components.

4.Compositionality: The ability of a system to combine various components to perform a task, where each component handles some part of the task to be executed.

5.Heterogeneity: Heterogeneity considers the diversity and dissimilarities in the units and components. Cyber physical systems are heterogenous distributed systems.

6.Autonomy: Being able to support (autonomous) reasoning, planning and decision- making via hardware, software, sensors and communication technology to increase a manufacturing system’s productivity and flexibility.

7.Predictability: property of a system to eliminate failures before they happen by sensing the situation.

8.Adaptability: the capability of system to reconfigure services and behavior due to change in external environment. uncertain conditions, and it possesses flexibility when it can adapt to changes in the external environment

9.(decentralized control)– Smart machines must have the appropriate level of intelligence to assess data quickly and in a decentralized fashion.

10.Interoperability: The ability of units to exchange and share information among them.

11.Cyber security: Data should be secured from cyber threats like privacy Intrusion, tampering, counterfeiting and other malicious Attacks.

12.Status: This is used to describe the present state of the activities that are being carried within the SMS. Asset self-awareness will also mean that the SMS should be able to know about its present state.

13.Human in loopUsing human expertise and knowledge for validation of certain tasks. Providing user interface for operators to interact with machines. Like HMI tools.

14.Dependability: Cyber-Physical Systems ideally should be dependable systems, which means essentially making them reliable, maintainable, available, safe, and secure.

15.Interfacing with legacy Systems: Integrating pre-existing designs (legacy systems) into new designs is a practical necessity for many Cyber physical systems applications.

16.Concurrency: the ability to run multiple computations or tasks at the same time.

17.Scalability: the potential of integrating additional systems to enhance production outputs, or computational power.

18.Digital mobility: it means being able to operate machines even remotely from far distances.

19.Industrie 4.0 compliant semantics: The exchange of information between two or more Industry 4.0 components requires explicitly specified semantics.

20. I4.0 compliant communication: capability of smart component to interact and exchange data with other components in the network.

1.4 Using a standard that defines a fixed set of properties for a smart machine

In this piece of work, the ZVEI standards have been considered for defining the properties of a smart machines. ZVEI which is a major manufacturing association in Germany, formulated the criteria for identifying a product as industry 4.0 ready product.

These criteria help providers in the market to decide which products can already be labelled as Industry 4.0-capable today. At the same time, companies can use these criteria as a guide for product development. For customers, the ZVEI definition provides clarity about the performance and features that Industry 4.0 products should provide. This ensures more transparency and security for the market as a whole. Consequently, such a standard also makes it clear what is not an Industry 4.0- compliant product. These criteria are derived from the standardized Reference Architecture Model Industry 4.0 (RAMI 4.0).

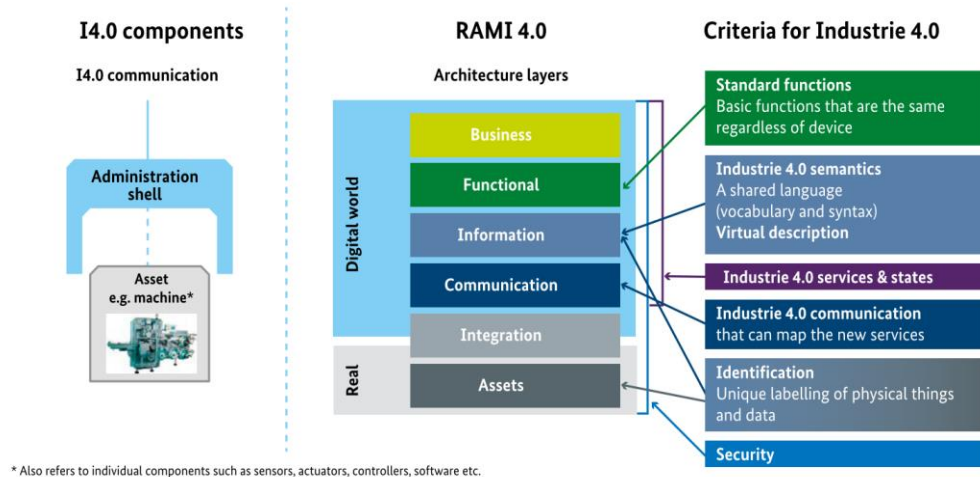


Figure 1.1: deriving criteria for industry 4.0 product from RAMI 4.0 architecture[8]

As per the standard, the product characteristics must qualify to their stated requirements of the characteristics. The lifecycle (L) is roughly divided into two phases of type (T) (development) and instance (I) (production, service). The degree of fulfillment (E) tells us the degree of importance where, (M) means mandatory (O) means optional and (N) indicates not relevant

The following tables provide an idea of the requirements stated by ZVEI to call a product to be industry 4.0 ready along with a smart product itself.

1.4.1 Identification[8]

Requirements	L	E	Product characteristics	Example: Nexo cordless WiFi nutrunner
Manufacturer independent identification of the asset with unique identifier (ID) attached to the product, electronically readable.	T	M	For 1) material number (electronic) in accordance with ISO 29002-55 or URI	1) Material number (electronic)
	I	M	For 2) serial number or unique ID For 3) manufacturer + serial number or unique ID With 2) and 3) electronically readable, for physical products via 2D code or RFID For 4) participant identification via IP network	2) QR code 3) QR code 4) Participant identification via TCP/UDP and IP network

Table1.1: requirements for property of Identification for a smart product

1.4.2 Industry 4.0 semantics[8]

Requirements	L	E	Product characteristics	Example: Nexo cordless WiFi nutrunner
<p>Standardized data in the form of attributes with cross-manufacturer unique identification and syntax for:</p> <ol style="list-style-type: none"> 1) Commercial data 2) Catalogue data 3) Technical data: mechanics, electronics, functionality, location, performance 4) Dynamic data 5) Data on the lifecycle of the product instance 	T	M	<p>For 2) catalogue data can be accessed online in an open standard 1–4) preferably ecl@ss, but also IEC CDD/W3C/ IEC 62832, IEC61360/ISO13584 and IEC61987-compliant data 2-3) Automation ML</p>	Yes, via QR code
	T	M	<p>For 2) and 5) catalogue data and data on the lifecycle of the product instance can be accessed online 3–5) preferably ecl@ss, but also IEC CDD/ W3C/ IEC 62832-compliant data.</p>	Yes, via QR code

Table1.2: requirements of industry 4.0 semantics for a smart product

1.4.3 Virtual description[8]

Requirements	L	E	Product characteristics	Example: Nexo cordless WiFi nutrunner
Virtual representation in I4.0-compliant semantic.	T	M	Customer-relevant information can be retrieved digitally based on type-identification (Product description, catalog, picture, technical features, datasheet, security properties, etc.) but further customer-relevant data are available in I4.0-compliant formats. Data on product types can also be transferred to public or private clouds (Administration shell via a type).	The product description, catalogue, image, technical features, datasheet etc. are available online
Virtual representation for the complete life cycle.				
Important properties of the physical component	I	M	Digital contact to service and to information regarding product support (including spare parts information) possible from the field. Representation of all production and service documents as well as data present and available internally in a transparent manner.	QR code leads directly to services and offers information on spare parts
information regarding the relation between Properties				
relations relevant for production and production process-relevant relationships between Industry 4.0 components	I	M		
formal description of relevant functions of the actual component and its processes				

Table1.3: requirements of Virtual description for a smart product

1.4.4 Industry 4.0 services and states[8]

Requirements	L	E	Product characteristics	Example: Nexo cordless WiFi nutrunner
Definition still open (service system)	T	O	Digital description of the device interface available	Openly described interfaces
General interface for loadable services and report of states. Necessary base services, which have to be supported and provided by an I40-product.	I	O	Information such as states, error messages, warnings etc. available via OPC-UA information model in accordance with an industry standard.	Data at the interface for all states are disclosed and can be requested.

Table1.4: requirements of Industry 4.0 services and states for a smart product

1.4.5 Standard functions [8]

Requirements	L	E	Product characteristics	Example: Nexo cordless WiFi nutrunner
Basic standardized functions, which can be executed manufacturer-independently and which provide same data in same functions. These basic functions serve as base for the functionality, on which every manufacturer can build their own extensions.	T	N	Functions described in Administration shell in format of I4.0 sub models.	First diagnosis and condition monitoring functions
	I	N	Functions implemented in Administration shell in format of I4.0 sub models.	Also monitoring of the process with diagnosis output

Table1.5: requirements of standard functions for a smart product

1.4.6 Security[8]

Requirement	L	E	Product characteristics	Example: Nexo cordless WiFi nutrunner
Minimum requirements for providing security functions	T	M	<p>A threat analysis has been executed.</p> <p>Appropriate security features were considered and publicly documented.</p> <p>Security by design.</p> <p>Security capabilities are described at the respective level (Authentication of the identifiers, user and role management, secure communication, logging of the security-relevant changes).</p>	<p>Security by design.</p> <p>Security capabilities are described at the respective level (Authentication of the identifiers, user and role management, secure communication, logging of the security-relevant changes).</p>
	I	M	<p>Available security capabilities are documented.</p> <p>Appropriate secure identities are available.</p>	Is discussed and documented in the customer projects
			<p>Security by design.</p> <p>Security capabilities are described at the respective level (Authentication of the identifiers, user and role management, secure communication, logging of the security-relevant changes).</p>	

Table 1.6: requirements of security for a smart product

1.4.7 Industry 4.0 communication[8]

requirements	L	E	Product characteristics	Example: Nexo cordless WiFi nutrunner
<p>Transfer of product data and data files for interpretation or simulation,</p> <p>for example; product data in standardized form.</p>	T	M	<p>Manufacturer makes data available/accessible online. The data should be relevant to customers and available/accessible with the assistance of identification/e.</p> <p>g. pdf via http(s) and URI</p>	Step Files, CAD drawings etc.

Product can be addressed via the network, supplies and accepts data, Plug & Produce via I4.0- compliant services	I		Administration shell of the product can be addressed (at any time) with the assistance of the identification online via TCP/UDP&IP with at least the information model from OPC-UA.	Yes, torque, rotary angle, tightening curve etc. can be read online
--	---	--	---	---

Table1.6: requirements of industry 4.0 compliant communication for a smart product

1.5 Current scenario of communication between devices in industries

The current industries hire system integrators to integrate each layer of their automation stack. The stacked automation layer approach raises issues of scalability and interoperability due to certain constraints such as compatibility issues because of different manufacturers and different formats of data-storages at each layer. Some original equipment manufacturers are able to provide solutions at each layer of automation stack. This means that the same manufacturer will provide for equipment, PLC's, SCADA systems and ERP systems. Such organizations claim to provide a fully operational vertical system integration, but the biggest challenge is that their products at each level are not compatible with those offered by other manufacturers. Industry 4.0 aims at industry wide integration of systems on a common platform through connected devices. Through IIoT information can be made available at any node in a network, in real-time. This would enable a link for efficient data transfer between machines and enterprise level solutions. MQTT and OPC-UA are some of the well-known IIoT solutions in industry.

In addition to network devices sharing data, this data is required to be in a certain standard which adds up to the interoperability of entities in the network.

The systems in industry 4.0 need to be collaborative and self-organizing to achieve their goals. This requires good communication among these systems. Therefore, it is important to facilitate a common language of interactions between these systems. In addition to data transfer in standard format, systems need to come up with autonomous decisions realized with effective communication between them. This project work uses FIPA-ACL as a standard language for these interactions and efforts are made to define a vocabulary for content specifications. This language constitutes of interaction patterns communication acts that mimic some of the human interactions.

1.6 Some benefits of interactive machines

The following tables are created in an attempt to understand the scenarios of real applications of the industry 4.0 ready products. The focus was on what are the reasons and benefits for communication between various shop floor entities. The following tables may act as a guide towards helping a manufacturer in identifying their needs from the benefits columns and invest in smart products accordingly.

Reason to communicate	outcomes	benefits
Retrieval of health parameters by a service. Ex: A health monitoring service wants to know whether the spindle temperature is in a specific range or not, so it asks for data from the temperature sensor.	Machine prognostics. (machine current health state and its likely failure can be known well in advance)	Repair costs go down. (we don't wait till the component fails by detecting failure possibilities using prognostics and thus we do less repairs)
	Control on energy consumption (a service that monitors energy consumption)	Costs related to energy usage are optimized.
Retrieving Information from part <i>in development stage</i> by a service. (i.e., when product is in virtual form, its manufacturing is yet to start). Ex: A process planning service asks for product design files.	Production planning Adaptability to changes in product design. (Production planning software can make a production plan for changes in product design or a completely new design)	Mass customization: Variety of products can be produced when separate production plan is available for each product.
MHS asks for optimal path of travel to a service. Ex. if an MHS is instructed to go from packaging section to inventory storage section, so it will ask a service responsible for traffic control to find the best suited path for it.	MHS travel path planning. A dedicated service that keeps a Realtime track of all the moving entities on the shop floor can provide for best path of minimum obstruction.	Minimization of material transport cost. (Smaller distance of travel will ensure more availability of MHS and Lesser energy consumption to transport the material to a specified location but via a shorter route)
Communication within various services. M2M communication itself can be a service which is being used by other services to communicate with machines. Ex. a sequencing service will require data from process planner(service) as well as it will require Machine	Very quick data transfer and decision making. (in an industry with considerably high number of machines, lot of time will be wasted in finding out the availability of required set of machines and further communicating this information manually, but in I4.0 scenario this time is comparatively negligible)	Better resource utilization Job and Machine idle time reduces as a consequence of reduction in time required for sequencing and scheduling

availability data which it will ask using the M2M communications(service).		
--	--	--

Table 1.7: Communication between machines and Cloud services/main system software/data repository

Reason to communicate	outcomes	benefits
A job arrives at a machine and it is unable to process the part and wants to send it to some another machine so it broadcasts.	Faster decision making Machines will do the bidding and based on those calculations the job will be assigned to that machine. This decision of assigning a job to another machine is very quick.	Minimization of production cost. (the machine selected in bidding will be based on calculations that minimize production cost)

Table 1.8: Communication between machine to another Machine

Reason to communicate	outcomes	benefits
Machine tells operator that it requires maintenance or repair. Example: there is a sudden blockage in coolant flow so the machine cannot process further. It will ask the operator to check the coolant flow.	Reduction in fault finding time. (In this case machine is able to identify the area of failure)	Machine downtimes during repairs will reduce.
Retrieving machine health parameters for maintenance by an expert. Ex. Expert operator wants to monitor vibrations of tool during a drilling process.	Tool degradation prognostics: (factors affecting excessive tool degradation can be monitored)	1. Reduced stock of spare parts. (tools) 2. Increased service life of parts. 3. Reduction in machine failures.

Table1.9: Communication between machine and operator

Reason to communicate	outcomes	benefits
<p>Production of a batch is finished and it is to be sent further.</p> <p>Ex: Machine tells the MHS to carry the finished pallet to next machine.</p>	<p>Shortest path selection:</p> <p>MHS decides a shortest path of travel when it is employed with the help of some service that manages shop floor traffic.</p>	<p>Lowering of material handling cost.</p>
<p>Machine broadcasts for knowing the availability of an MHS for transport of a pallet.</p> <p>Best suited MHS is selected based on its capacity, availability and presence in the vicinity.</p>	<p>Time saving in MHS finding:</p> <p>Ex: in case of large shop floors with multiple automated MHS, time will be required to manually find the idle MHS and instruct it for transportation of certain material.</p>	<p>Optimization of material transport cost.</p> <p>(on the basis of bidding the best MHS will be selected for transporting the material based on cost optimization)</p>

Table 1.10: Communication between machine and MHS

Reason to communicate	outcomes	benefits
<p>Inventory storage system asks for requirement of local storage near the machine.</p> <p>Ex. If a machine requires material pallets at some intervals, then inventory storage system asks the machine for these pallets at these intervals.</p>	<p>Lower machine idle time</p> <p>Pallets made available to the machine just in time.</p>	<p>Higher machine productivity.</p> <p>(machine productivity increases due to lower idle Time)</p>

Table 1.11: Communication between machine and inventory storage system

Reason to communicate	outcomes	benefits
<p>(Job to machine) how much progress has been achieved on machining.</p> <p>Ex. Job has a repository which continuously gets updated after each process done on job. This repository stores updates on job status.</p>	<p>Process monitoring.</p> <p>Ex: with the continuous update on machining process, a service or an operator can keep a watch on proper execution of processes on the part.</p>	<p>a smaller number of defective products produced.</p>
<p>How fast the machine can carry out a particular process on the job.</p> <p>Ex. In case of excessive demands with lower customer lead time the job will try to find a machine that will execute the part the fastest.</p>	<p>Flexibility of throughput and Adaptability to variable demand conditions.</p> <p>Ex. In case of higher demands a greater number of machines will be employed in production of part that is in higher demand and production of other parts can halt.</p>	<p>Conformity to produce and deliver within customer lead time.</p> <p>(The throughput can be adjusted according to demands which lead to availability of finished goods at desired time.)</p>

Table 1.12: Communication between machine and job

The above-mentioned examples require machines to have some sort of intelligence, that would facilitate decision making in an autonomous way.

1.7 Requirement of standards for communication

The social capabilities of smart machines are a key factor for self-organized systems, which demands intelligent interactions among the entities in an industry. This can be efficiently achieved by providing a common platform and a common language for communication. Semantics related to this language must be known by the entities on this platform. Thus, a standard can enable machines to interact, independent of manufacturer which play an important role in scalability and industry wide integration of systems.

Chapter 2: Literature review

2.1 What is industry 4.0

The ideas of industry 4.0 has its manifestation in various literatures with different names. Smart manufacturing, Factories of the future and connected industry are some of the common terms that are used in similar context to that of Industry 4.0. Various manufacturing associations and literatures have tried to give a definition to industry 4.0. In the literature [1] there are two definitions given by NIST and SMLC. the National Institute of Standards and Technology (NIST) smart manufacturing are systems that are “fully-integrated, collaborative manufacturing systems that respond in real time to meet changing demands and conditions in the factory, in the supply network, and in customer needs”. The SMLC definition states that "Smart Manufacturing is the ability to solve existing and future problems via an open infrastructure that allows solutions to be implemented at the speed of business while creating advantaged value."

2.2 Interoperability in context of Industry 4.0

Interoperability in general terms, as defined in [1] is *“the ability of two or more entities to interact and cooperate”*. [9] states that interoperability is *“the ability of one system to receive and process intelligible information of mutual interest transmitted by another system”*. [1] also explains about the two types of interoperability, namely syntactic and semantic interoperability. Syntactic interoperability only considers the format of the data transfer whereas semantic interoperability deals with the meaning of the expressions in the message content.

2.3 Challenges in Implementation of Interoperability

The Manufacturing Interoperability Program at NIST (the National Institute of Standards and Technology) list several factors that impact the effectiveness of interoperability as cited in [1] :

- Transfer of data between systems that may be similar or dissimilar (commercially).
- Transfer of data between software made by the same vendor (or creator) but having different versions on the systems.
- Compatibility between different versions of software (newer and older versions).
- Misinterpretation of terminology used or in the understanding of the terminology used for exchange of data or information.
- The use of non-standardized documentation on which the exchange of data is processed or formatted.
- Not testing the applications that are deemed conformant, due to the lack of means to do so between systems.

Other barriers to interoperability include inconsistent data formats or standards, connectivity in the IoT realm, and the wide variety of commercially available products.

2.4 How standardization in industry 4.0 can help enhancing interoperability

As per [10], standardization plays a key role for adoption of Industry 4.0. Industry 4.0 requires a common language and common security for the following reasons.

- 1) common language allows all components to be related and enables them to exchange information through the same vocabulary, syntax, semantics, formats, physical interfaces, communication protocols, interoperability, management platforms, among others.
- 2) common language to design solutions based on Industry 4.0 standards, which prevents each supplier from using their own standards and facilitate training and specialized technical support.
- 3) Common security for protection of business information, privacy of people and all the information that is produced, transmitted and stored.

A common language dictates the need of a standard for communication. In this piece of work, efforts were made to identify and implement a standard for communication with the use of an already defined open standard named FIPA-ACL.

2.5 Need of multi-agent systems (MAS) approach

[11] explains the criteria setup by EURESCOM, by which one can judge whether there is a need to use multi-agent systems or not. The MAS approach should be used when:

- When complex/diverse types of communication are required
- When the system must perform well in situations where it is not practical/ possible to specify its behavior on a case-by-case basis
- When negotiation, cooperation, and competition between entities is involved
- When the system must act autonomously
- When high modularity is required (for when the system is expected to change).

Chapter 3: how to enable machines to interact

3.1 Basics of device interactions

For basic transfer of messages between entities, it is essential that these entities are connected via some form of network. Secondly these entities must be identifiable so that communication can be possible with them. There are broadly 2 kinds of architectures that are adopted by networking and IIoT software, peer to peer interactions and client server interactions. The peer-to-peer interaction architecture has problems in scalability, the same factor for which client-server-based architecture is stable while scaling.

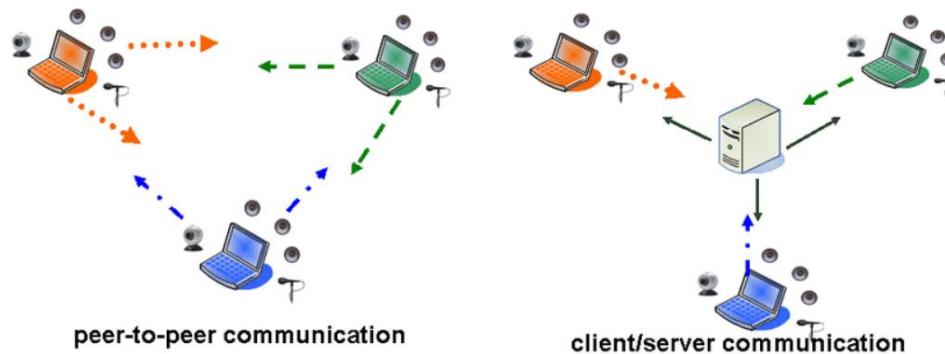


Figure 3.1: basic types of network model [google images]

The entities in a network must also adhere to some specific format(s) of data transfer that both the sides should understand. And also, for complex communications, the contents of the messages should be such that both the sides should understand. The content part of the messages thus can be standardized and each entity may be programmed to adhere to these standards. Thus, it is important to determine first, what will be the network architecture, messaging format, messaging language and the message transport mechanism.

3.2 Messaging Format

Computer systems may vary in their hardware architecture, OS, addressing mechanisms. Internal binary representations of data also vary accordingly in every environment. Storing and exchanging data between such varying environments requires a platform-and-language-neutral data format that all systems understand. Messaging format refers to the structure of message or the format in which message content is expressed.

Extensible Markup Language (XML) and JavaScript Object Notation (JSON) are widely used tools for the presentation of arbitrary data structures, which are already mature for data transfer of connected machines. This provides common frameworks to allow data to be shared and reused across applications with the standards to facilitate the data formats and exchange protocols. In this project work, the messages being sent between machines are basically serialized JSON format files.

JSON stands for JavaScript Object Notation. It is a lightweight format for storing and transporting data and is often used when data is sent from a server to a web page. JSON is "self-describing" and easy to understand as it is in human readable as well as machine readable form.


```
{
  "employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
  ]
}
```

Fig 3.2: representation of JSON format

3.3 Message transport mechanism

Application layer protocols such as SMTP, FTP, HTTP directly interact with the web applications, the data to be transferred is broken down into data packets by the transport layer protocols that includes TCP and UDP. The internet protocol (IP) assigns origin and destination IP address to each packet. The network layer handles MAC addressing and conversion of information into electrical impulses that gets transported using physical hardware protocols.

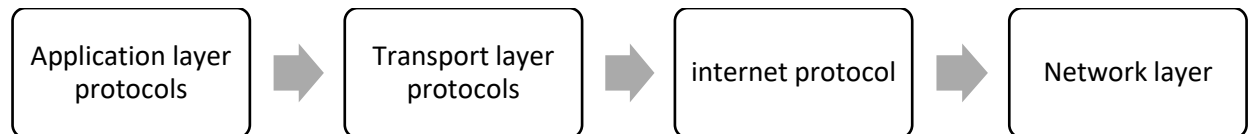


Fig 3.3: basic TCP/IP communication model

This project uses web-sockets for communicating among devices, which is based on the TCP/IP model.

3.4 Messaging Language

The messages that get shared between the agents are written in certain format and the content of this message is expressed in a particular language.

FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. FIPA, the standards organization for agents and multi-agent systems was officially accepted by the IEEE as its eleventh standards committee in 2005.

FIPA organization developed a machine-to-machine communication standard that is known as “agent communication language” or FIPA-ACL. It is an open standard that was created for an agent-based architecture for machines to interact with each other. The details of this language are mentioned further in this document.

3.5 Why multi agent systems

FIPA-ACL was initially defined for an agent-based structure. But since it is a language, it can be used by connected devices irrespective of network structure and interaction capabilities. But such a language will find its use much more in a network that can promote peer-to-peer interactions rather than publish/subscribe type interactions. Following are some of the reasons why multi agent systems architecture is suitable for demonstration of use of this language.

1. **Parallel computation:** Having multiple agents could speed up a system's operation by providing a method for parallel computation. For instance, a domain that is easily broken into components--several independent tasks that can be handled by separate agents--could benefit from MAS. Furthermore, the parallelism of MAS can help deal with limitations imposed by time-bounded reasoning requirements.

2. **Scalability:** Since they are inherently modular, it is much easier to add new agents to a multiagent system than it is to add new capabilities to a monolithic system. Systems whose capabilities and parameters are likely to need to change over time or across agents can also benefit from this advantage of MAS.

3. **simpler programming:** Rather than tackling the whole task with a centralized agent, programmers can identify subtasks and assign control of those subtasks to different agents. The difficult problem of splitting a single agent's time among different parts of a task solves itself.

4. **Supports intelligent interactions:** The multi agent systems support intelligent interactions amongst its agents through some form of interaction patterns and a support for incorporating a standard language for these interactions.

Chapter 4: messaging language

In essence a language of communication in context of machine interactions is a set of words in a message which are arranged in a particular structure, used to understand the context of a dialogue.

If the message just includes an expression, then this expression must contain words and characters that have well defined semantics and ontology, so that the receiver can successfully understand the meaning of that phrase. There are two major messaging languages that were developed and standardized for agent based communications KQML and FIPA-ACL. Their descriptions are given below.

4.1 Knowledge Query and manipulation language (KQML)

KQML is an 'outer language' that defines an envelope format for messages that get transferred between agents[12]. KQML defines various 'communicative verbs' called as performatives (E.g., Ask-if, perform, tell, reply etc.) The performative along with the message content will constitute a speech act.

example:

Performative: **Request**
content: "the door is closed"
speech act: "please close the door"



Fig : constituents of a speech act

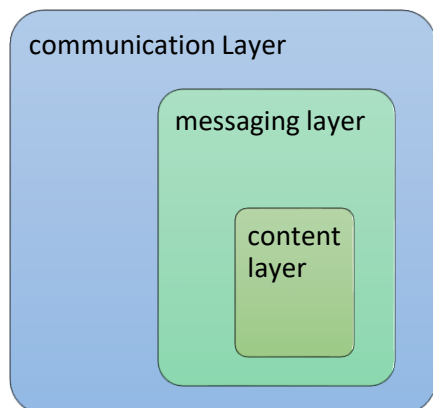


Fig : KQML layered structure

```
(PACKAGE :FROM ap001
:TO ap002
:ID DVL-f001-111791.10122291
:COMM sync
:CONTENT
(MSG
:TYPE query
:QUALIFIERS (:NUMBER-ANSWERS 1)
:CONTENT-LANGUAGE KIF
:CONTENT (color snow _C)))
```

communication layer

message layer

content layer

The KQML message structure consists of the following layers:

Content Layer: KQML has no restrictions on the content language, one can use KRSL, KIF, LOOM or any other semantic language.

KQML Message Layer: The message layer is used to encode a message that one application would like to have transmitted to another application.

It should primarily consist of:

- Message type (performative)
- Content language (e.g., KIF)
- Content ontology (e.g., OWL)
- Content topic
- Content itself

```
(DCL
  :TYPE assert
  :DIRECTION export
  :MSG
    (MSG
      :TYPE assert
      :CONTENT-LANGUAGE KIF
      :CONTENT-ONTOLOGY (blocksWorld)
      :CONTENT-TOPIC (physical-properties)
      :CONTENT (color ?X ?Y)))
```

Fig Sample of a declaration message [13]

KQML Communication Layer: At the communication layer, agents exchange packages. A package is a wrapper around a message which specifies some communication attributes, such as a specification of the sender and recipients. A package is represented as a list of keyword arguments. Possible keyword arguments are:

- Sender name (FROM:)
- Receiver name (TO:)
- Package ID
- Communication type (synchronous or asynchronous)
- The message

KQML is independent of:

- the transport mechanism (tcp/ip, cobra etc.)
- the content language (KIF, SQL etc.)
- the ontology assumed by the content.

4.2 FIPA-ACL (Foundation for Intelligent Physical Agents - agent communication language)

FIPA-ACL defines a structure for messaging, and the expressions can be written in the content part of a FIPA-ACL message. It is very much similar to KQML

FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. FIPA is a well-known standard that explicitly specifies the interaction patterns and messaging language structure (ACL) for interactions amongst autonomous decentralized agent-based architecture

FIPA-ACL consists of 20 performatives (communication acts). Semantics of these performatives are given with respect to a formal semantic language (FIPA-SL). FIPA-ACL structure is similar to KQML.

```
{
  'protocol' : 'request_interaction_protocol',
  'performative' : 'request',
  'sender' : 'job_manager',
  'receiver' : 'machine_1',
  'type' : 'request_due_date_change',
  'content' : '[12,22,07-20-2021,3,2,machine-Agent 2]',
  'conversation_id' : 'RIP-Cx1e',
}
```

Fig 3.7: FIPA-ACL message

4.3 Concept of a performative

The concept of performative or a communication-act has its origin in the speech act theory. According to a speech act theory, the utterance of words for communicating something from one person to another can be perceived as an act of communication.

For example, if person-A says to person-B, “please close the door.”. Here we can see that there has been an act (uttering those words) for a communication and also the nature of this act is a request (as the word “please” is use). Therefore, we can say that person-A used a “request” communication-act for asking person-B to close the door. Had the sentence been “close the door!”, the communication act for this sentence can be perceived as an “order” rather than a request.

The various types of communication-acts are called as performatives. FIPA has recognized 22 such performatives that can be used by agents in an agent-based system to communicate. They are listed down below.

4.4 Description of performatives

FIPA-ACL consists of 22 performatives as enlisted below

1. Accept-proposal: The action of accepting a previously submitted propose to perform an action.
2. Agree- The action of agreeing to perform a requested action made by another agent. Agent will carry it out.
3. Cancel- Agent wants to cancel a previous request.
4. Call-for-proposal- Agent issues a call for proposals. It contains the actions to be carried out and any other terms of the agreement.
5. Confirm- The sender confirms to the receiver the truth of the content. The sender initially believed that the receiver was unsure about it.
6. Disconfirm- The sender confirms to the receiver the falsity of the content.
7. Failure- Tell the other agent that a previously requested action failed.
8. Inform- Tell another agent something. The sender must believe in the truth of the statement. Most used performative.
9. Inform-if- Used as content of request to ask another agent to tell us is a statement is true or false.
10. Inform-ref- Like inform-if but asks for the value of the expression.
11. Not-understood - Sent when the agent did not understand the message.
12. Propagate - Asks another agent so forward this same propagate message to others.
13. Propose - Used as a response to a call-for-proposal. Agent proposes a deal.
14. Proxy - The sender wants the receiver to select target agents denoted by a given description and to send an embedded message to them.
15. Query-If - The action of asking another agent whether or not a given proposition is true.
16. Query-ref - The action of asking another agent for the object referred to by a referential expression.
17. Refuse - The action of refusing to perform a given action, and explaining the reason for the refusal.
18. Reject-proposal - The action of rejecting a proposal to perform some action during a negotiation.
19. Request - The sender requests the receiver to perform some action. Usually, to request the receiver to perform another communicative act.
20. Request-when - The sender wants the receiver to perform some action when some given proposition becomes true.

21. Request-whenever - The sender wants the receiver to perform some action as soon as some proposition becomes true and thereafter each time the proposition becomes true again.
22. Subscribe - The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.

Table 4.1: all FIPA-ACL performatives

4.5 Concept of Interaction protocol

According to FIPA-ACL standard, an interaction among machines is basically the exchange of performatives between machines or the Agents. But combination of some performatives can result into different types of conversations. A predefined pattern of performative exchanges is called as an interaction protocol. These interaction protocols have also been standardized by FIPA.

For example, if agent-A sends a request performative to agent-B, agent-B now has 21 other performatives to give a reply, this can confuse agent-B as to which performative to send back, therefore if agent-A also specifies that it has sent a message i.e. A request performative under request-interaction protocol, then agent-B's choice of reply performatives narrows down to 3, as there are only 3 kinds of reply performatives to a 'request' performative sent under 'request interaction protocol'. Further the final performative chosen to reply out of these 3 will be the one decided as the message gets processed further. Additionally, an interaction protocol is important to understand and track the start, end and current status of a conversation.

4.6 Various interaction protocols

FIPA-ACL is a language created for agent interactions. But due to agents being used in various sectors such as finance, manufacturing, marketing etc. Therefore the choice of performatives and interaction protocols has been kept vast. In this project work however, only those interaction protocols have been considered that would suffice the needs of interaction capabilities of machines in manufacturing industry. Following is an explanation to those particular interaction protocols. The description and working of these protocols is explained in [14]

4.6.1 Request Interaction Protocol

The FIPA Request Interaction Protocol (IP) allows one agent to request another to perform some action.

Explanation of the Protocol Flow

Request interaction protocol starts with a request performative. The type of request and specifications are identified from the message. The expected responses are 'refuse' performative to refuse the request and 'agree' performative to agree upon further response to request. If agreed, the agent that sends an agree performative is expected to inform with confirmation of completion of action or with result. In this case the conversation ends with an 'inform' performative.

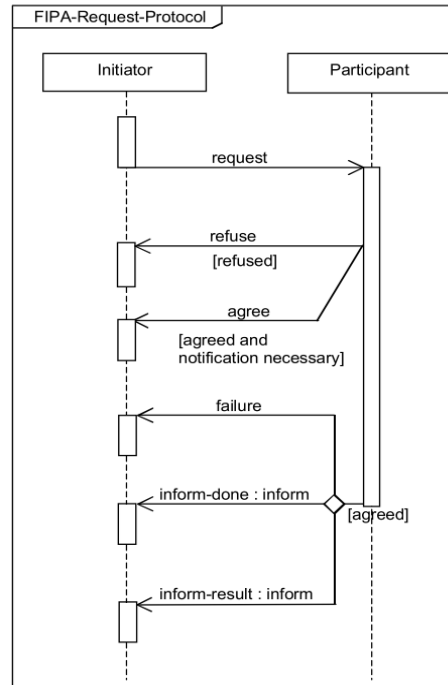


Fig 4.1: Interaction pattern of FIPA request protocols[15]

4.6.2 FIPA Contract Net Interaction Protocol

FIPA Contract net protocol is an elementary level interaction protocol for auction. The interaction can take place between an initiator and many responders.

Explanation of the Protocol Flow

The initiator sends a 'call for proposal' to the designated target agents basically asking each agent to give a proposal in accordance to specifications of the proposal. These target agents either respond with a 'propose' performative or a 'refuse' performative. The initiator waits till a specific time to accumulate all the proposals and processes them to finalize on the best proposal. The agent that had sent best proposal is sent 'accept-proposal' performative and rest of the agents are sent a 'reject-proposal' performative. The Agent chosen through this process has to inform the results or notify the completion of task to the initiator agent.

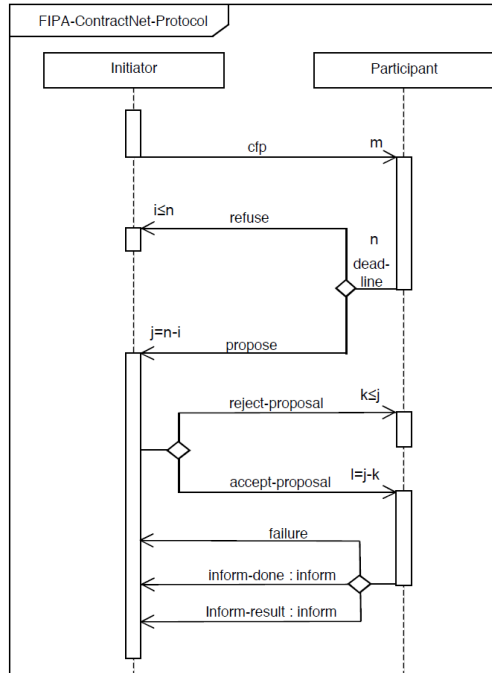


Fig 4.2: FIPA contract-net interaction protocol flow[16]

4.6.3 FIPA Subscribe Interaction Protocol

The FIPA Subscribe Interaction Protocol (IP) allows an agent to request a receiving agent to perform an action on subscription and subsequently when the referenced object changes.

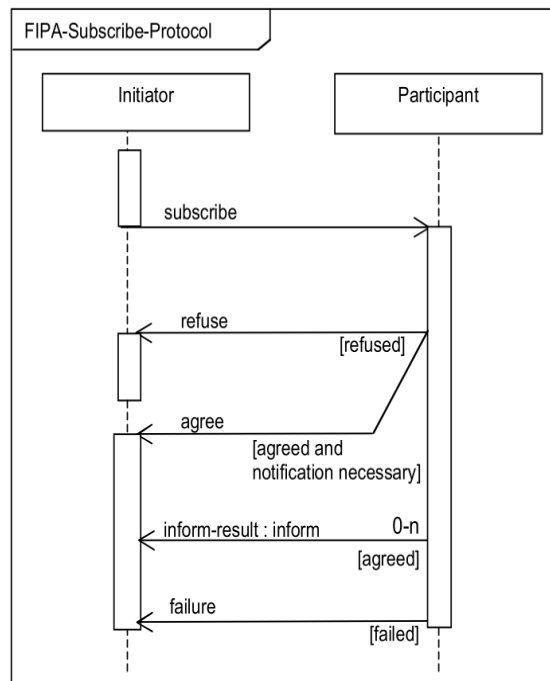


Fig 4.3: FIPA Subscribe protocol flow[17]

Explanation of the Protocol Flow

The subscribing process is simple. The 'subscribe' performative is sent to the agent that handles certain topics for subscription. The specific topic of interest may be mentioned in the content specifications. The response expected from the recipient agent is a 'refuse' or 'agree' performatives. The further updates on these topics are sent to the subscribers. To unsubscribe, "cancel meta protocol" will be used.

4.6.4 FIPA Request-When Interaction Protocol

The FIPA Request When Interaction Protocol (IP) allows an agent to request that the receiver perform some action at the time a given precondition becomes true. This IP provides a framework for the request-when communicative act

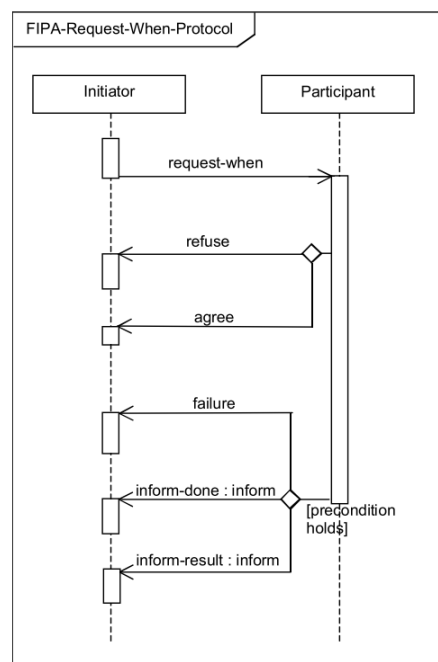


Fig 4.4: FIPA request-when interaction protocol flow[15]

Explanation of the Protocol Flow

The protocol flow is almost same as FIPA Request interaction protocol. The request when performative is sent with precondition in the specifications. The receiver agent will reply back with an 'agree' performative, to inform when the precondition becomes true. Or else it may reject the request with a 'Refuse' performative. Later if the request was agreed and the precondition becomes true, the participant agent will send an 'inform' performative and notify about completion of task.

4.6.5 FIPA Query Interaction Protocol

The query interaction protocol is used to basically confirm whether certain preposition is true or not.

Explanation of the Protocol Flow

The initiation of protocol is done with “query-if” or “query-ref” performative. The query-if performative is used to ask about the trueness of a preposition in the content expression. Query-ref performative is used when there is a query in reference to a particular object of interest. The rest of protocol flow is similar to the request interaction protocol. Where the participant agrees or refuses to respond. If agreed, it is followed by an inform performative.

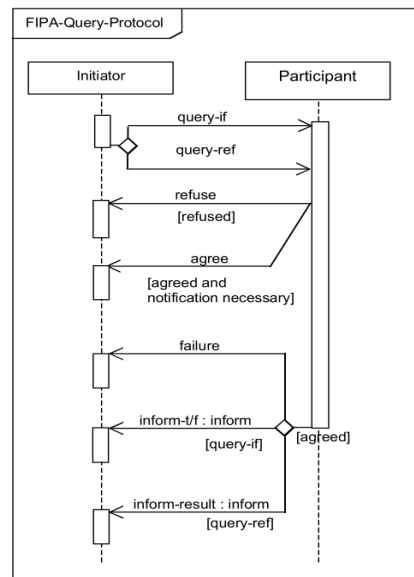


Fig 4.5: FIPA query protocol flow[18]

4.6.6 FIPA Propose Interaction Protocol

The FIPA Propose Interaction Protocol (IP) allows an agent to propose to receiving agents that the initiator will do the actions described in the propose communicative act when the receiving agent accepts the proposal.

Explanation of the Interaction Protocol Flow

Propose interaction protocol is basically a shortened version of contract net interaction protocol, where the recipient of the message or proposal is not more than 1.

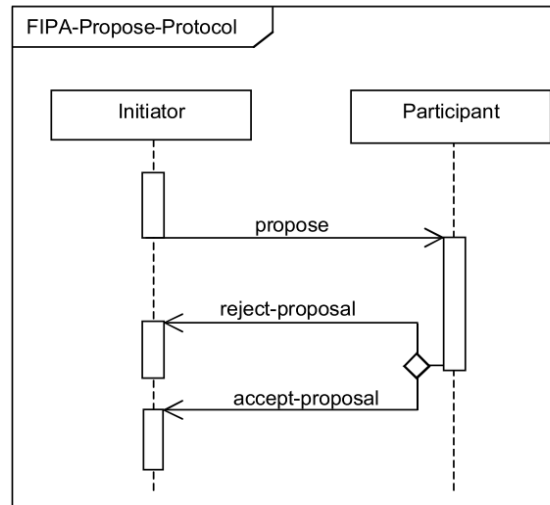


Fig 4.6: FIPA Propose Protocol flow[19]

4.6.7 FIPA cancel meta protocol

The cancel meta protocol is used as a sub-protocol to terminate any ongoing interaction protocol. The agent that wants to cancel the interaction just sends a 'cancel' performative. The type parameter may include the particular performative to cancel, here the conversation id plays an important role to identify and terminate an existing interaction. This protocol finds its use mostly in subscribe protocol, request-when protocol, and contract net protocol. It can be used to terminate other interaction protocols as well if they have allowed a time limit to respond in 'reply-by' parameter.

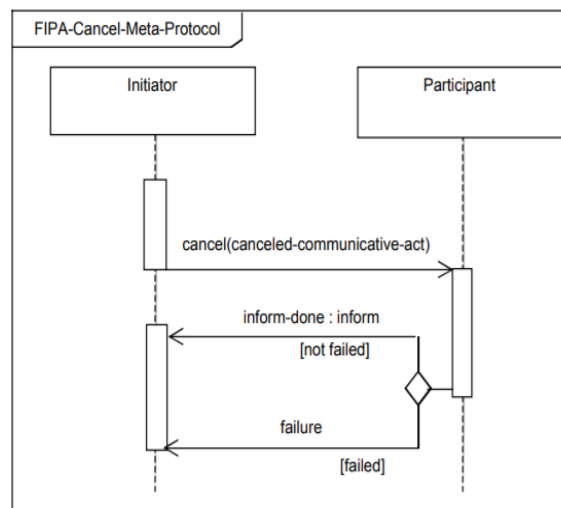


Fig 4.7: FIPA Cancel-Meta Protocol flow

4.6.8 FIPA recruiting interaction protocol and FIPA broking interaction protocol.

The recruiting protocol and broking protocol are basically proxied interaction protocols. These two protocols ask some other agent (a proxy agent) to perform an interaction for them.

The major difference between these two protocols lies in the initiator agent's knowledge on who to send the message. If the initiator agent already has the knowledge to figure out who are its target agents, in that case the proxy agent is sent a list of target agents. However, when agent does not have sufficient knowledge to figure out the target agents for the interaction, it will ask a "broker agent" to do the talking for it, as this broker can figure out who the target agents can be. So, the proxied agent becomes a broker agent.

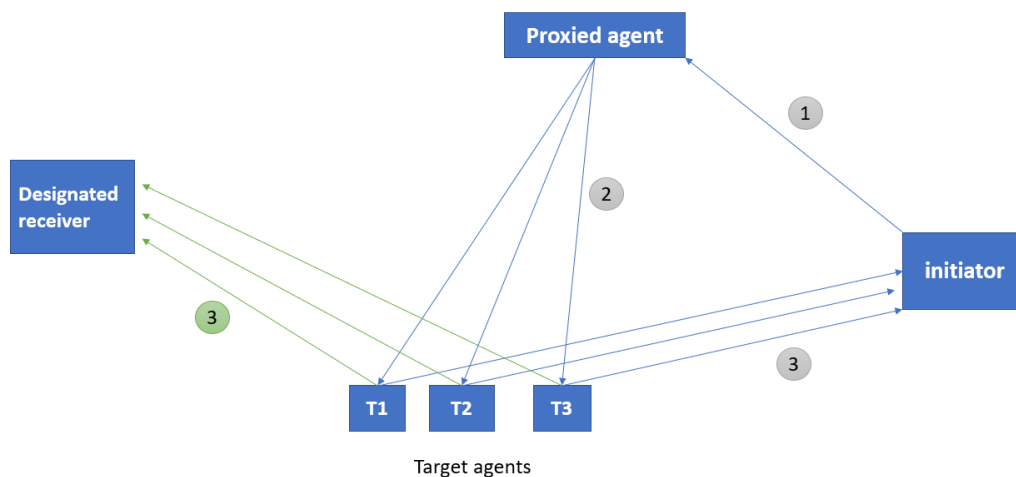


Fig 4.8: Proxy interaction

As shown in the above figure, the initiator agent asks a proxy agent to carry on certain interactions. In the figure; (1) represents sending a 'proxy' performative. If recruiting interaction protocol is used then message will include list of target agents and performative to be sent to them, along with the content. If broking interaction protocol is used then no such list of target agents is sent, but an expression that specifies certain constraints is sent that helps the broker understand how to select the target agents. (2) illustrates the interaction of broker with target agents in the performative that was stated by the initiator. And (3) depicts the response of this interaction, the responses can be either sent back to the initiator or to certain designated receiver as specified by initiator.

4.7 Concept of performative types

Performative types do not have a mention into FIPA literatures. These have been introduced in this project. A performative type is basically an additional information for the agent to understand how to handle the content part of the FIPA message.

For example, if Agent-A sends a message with request-performative to agent-B, now to understand the content part of the message, agent-B will look at the performative-type mentioned in the message, let us say the type is “request_job_info”. Now agent-B has a function that would respond to this performative type and hence agent-B was able to identify and send message content to this function. This is the way the performative types become useful.

The FIPA-ACL performatives are more generalized. As an example, ‘request’ performative tells that there is a request being made, but the context and specifications of request are to be understood from the type and content of the message. For this purpose, there is a need of explicitly specified types that each machine understands. The types are linked to a specific performative only. Below table gives a list of some of the types that some performatives have.

It should be noted that all the specified performatives do not need a type, especially when they occur as a last message in a conversation such as refuse, agree, etc. And also, some of these tags may repeat for different performatives. This simply gives a context of using the type in relation to that performative.

Example: “failure in transit” type occurs in request-when as well as call for proposal. But the uses for the type in both cases is different. A machine may tell the material handling entity to inform when there is failure in transit, so in this case machine uses request-when performative and failure in transit as a type. But in some other case where an MHS that fails, wants some other MHS to transport the load, so it uses a call for proposal performative with failure in transit type.

Performative	Types
Request	Request to transport, Request access, Request info, Request to load, Request optimum path, request for process plan, etc.
Request when	Request to load, Request to unload, failure in transit
Call for proposal	Job processing failed, CFP for transport, new job auction, failure in transit
Cancel	Cancel (performative name) ex. Cancel_request, cancel_request_when etc.
Inform	Inform-result (content not empty), inform-done (content empty)
Subscribe	Subscribe_common_topics, subscribe_topic (to subscribe a particular topic)

Table 4.2: examples of performative types

4.8 FIPA-ACL message structure

A FIPA ACL message contains a set of one or more message parameters. Precisely which parameters are needed for effective agent communication will vary according to the situation; the only parameter that is mandatory in all ACL messages is the performative, although it is expected that most ACL messages will also contain sender, receiver and content parameters.

If an agent does not recognize or is unable to process one or more of the parameters or parameter values, it can reply with the appropriate not-understood message.

The following is a table of all possible message parameters that together constitute a FIPA-ACL message.

Parameter	Description	Importance
protocol	Particular interaction protocol	C
Performative	Request, call_for_proposal, inform, etc.	C
Sender	Identifier of sender	C
Receiver	Identifier of receiver	C
Reply_to	3 rd entity identifier (when sender wants receiver to reply to some other entity, not back to the sender)	
type	tag/Pointer to function	C
In_reply_to_type	'type' of prior message	
Content	# content part	C
Language	Language of content expressions	
Encoding	Format of content ex. JSON	C
Conversation_id	To keep a track of conversation.	
Ontology	Content ontology	
Reply_by	Reply within a particular amount of time	

Table 4.3: FIPA message structure (c = compulsory)

```
{
  'protocol' : 'request_interaction_protocol',
  'performative' : 'request',
  'sender' : 'job_manager',
  'receiver' : 'machine_1',
  'type' : 'request_due_date_change',
  'content' : '[12,22,07-20-2021,3,2,machine-Agent 2]',
  'conversation_id' : 'RIP-Cx1e',
}
```

Fig4.9: outer envelope syntax for the the FIPA-ACL message (json format)

4.8.1 Description of parameters

Performative: Denotes the type of the communicative act of the ACL message

Sender: Denotes the identity of the sender of the message, that is, the name of the agent of the communicative act.

Receiver: Denotes the identity of the intended recipients of the message.

Reply-to: This parameter indicates that subsequent messages in this conversation thread are to be directed to the agent named in the reply-to parameter, instead of to the agent named in the sender parameter.

Content: Denotes the content of the message; equivalently denotes the object of the action. The meaning of the content of any ACL message is intended to be interpreted by the receiver of the message. This is particularly relevant for instance when referring to referential expressions, whose interpretation might be different for the sender and the receiver.

Language: Denotes the language in which the content parameter is expressed.

Encoding: Denotes the specific encoding of the content language expression.

Ontology: Denotes the ontology(s) used to give a meaning to the symbols in the content expression.

Protocol: Denotes the interaction protocol that the sending agent is employing with this ACL message.

Conversation-id: Introduces an expression (a conversation identifier) which is used to identify the ongoing sequence of communicative acts that together form a conversation.

Type: type refers to a performative type or a particular tag or a pointer to a function that has to run when this tag is received.

In-reply-to-type: the 'Type' of previously sent message to which this message is a reply. This parameter will be blank when first message is sent at the start of an interaction protocol.

Reply-by: This can be thought of as a deadline to reply, and it denotes a time and/or date expression which indicates the latest time by which the sending agent would like to receive a reply.

4.9 Case example for how FIPA interactions take place

The following example explains how FIPA-ACL contract-net protocols is used for interactions within machines. This is a basic example for an auction.

Case description: let us consider a situation where there are machines as well as material handler systems in a network. In this particular case we will consider machine agents M1 and M2, and material handler system agents MHS1, MHS2 and MHS3.

The trigger for this interaction is that, machine (M1) has finished processing of the jobs and decides which machine should process the job further. Now machine(M1) has to decide, how the transport of material should take place.

At the first stage of interaction, conversation using contract-net interaction protocol is initiated and the machine M1 acquires a list of nearby material handling AGV's (MHS1, MHS2 and MHS3). These AGV's are sent a call for proposal with a 'cfp_to_transport' performative-type.

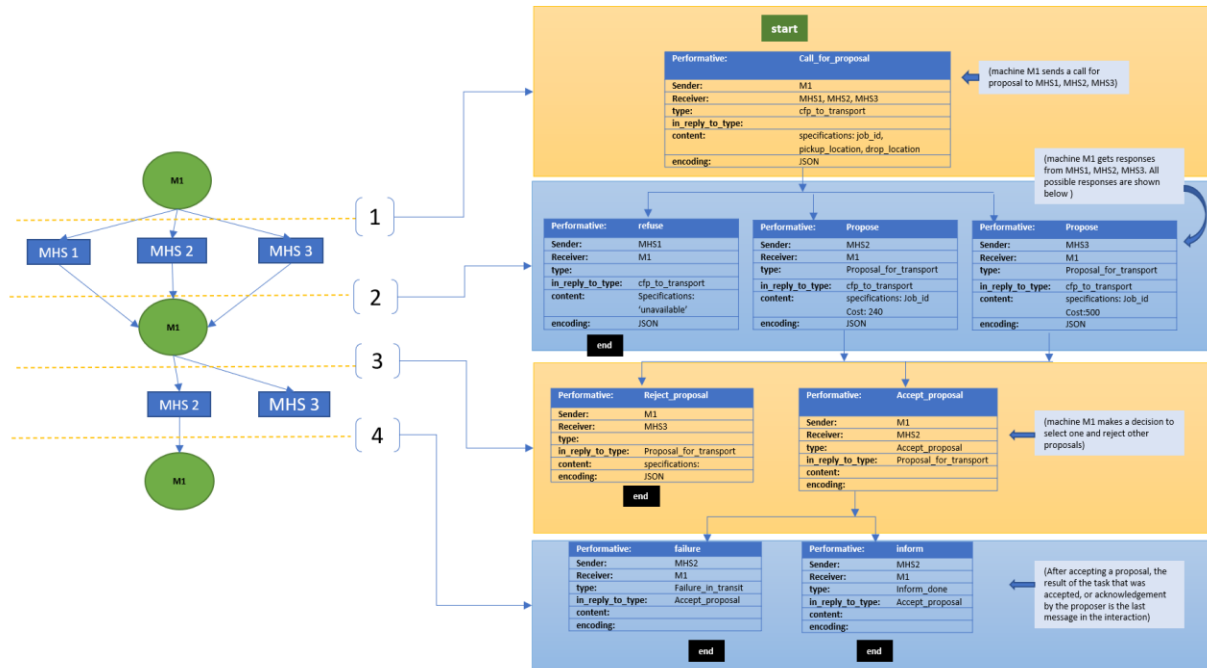


Fig 4.10 : example for contract-net interaction protocol

At the second stage of interaction, based on their current location and specifications of pickup and drop locations, each AGV will evaluate a cost and create a FIPA message with 'proposal_to_transport' performative-type, and a propose performative, and send this message to machine-1. The MHS1 refuses to propose so it sends a refuse performative with some reason and other two MHS propose a cost.

The third stage of interaction Machine M1 decides whose proposal was the best and therefore which particular proposal to accept based on cost, proposal of MHS3 is rejected and that of MHS2 is accepted.

At the fourth and last stage of interaction, MHS 2 fails to deliver or reach the pickup within time, it will notify a 'failure'. Else after completion of task the MHS2 will respond with inform performative with 'inform_done' type.

Chapter 5: A FIPA compliant agent-based system architecture

An agent is an entity that can make autonomous decisions in a decentralized way. they consist of interaction capabilities with the other agents and are aware of their surroundings. The Multi agent platform structure considered, consists of a central server and various agents that may dynamically register and deregister with this server. There is a central server that facilitate messaging amongst Agents.

5.1 The proposed multi-agent system Architecture

There are two ways of interactions in which clients communicate. Direct peer-to-peer communication and client-server-based communication. This particular Agent-based system will make use of the client-server-based model of communication. Each agent will communicate to another agent via server. This is because the client-server model is suitable for an extensive network, i.e. scalable, and there will be some level of centralized agent management included with the server. As shown in the following figure, all agents communicate with each other via the server. The details of each component of the architecture are explained in the subsequent chapters.

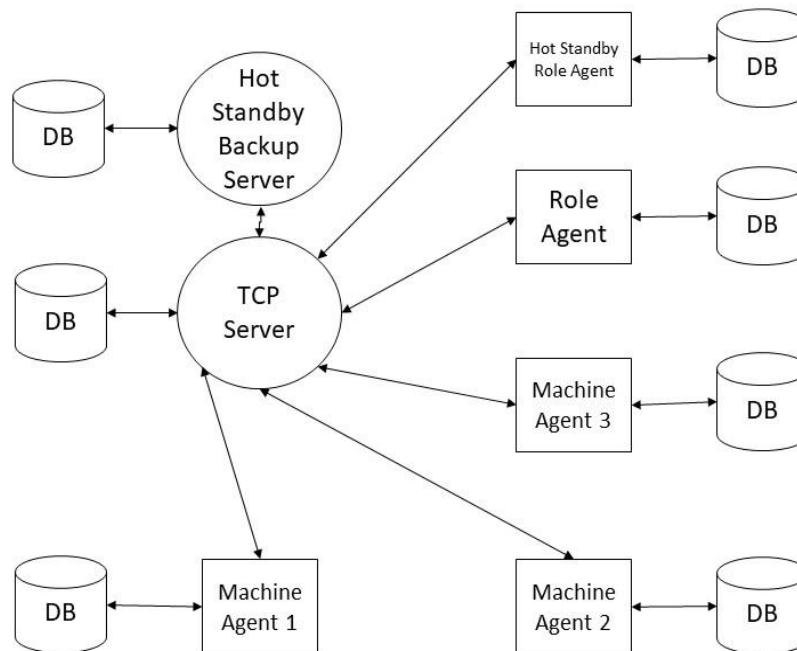


Fig 5.1: Proposed Multiagent Architecture

5.2 Web-sockets based client to server communication

A network protocol ensures the connectivity and transfer of messages between server and Agents. WebSocket protocol is a standard computer communication protocol which provides a full-duplex communication channel over a single TCP connection. TCP is the transport protocol for packets between these sockets which reside in each agent. The connectivity can be over Wi-Fi or through LAN cables.

5.3 Agent description File

The agent description file or agent file consists of agent name, Agent role and abilities of understanding the performative types. As shown in the following figure, “request_to_load_task” is a type (performative-type of request performative) input specifications means that the content part requires that particular specification mentioned. The output specifications means that the outputs of the functions that handle these performative types should have those particular specifications mentioned in content part.

Sending ability enabled means the agent can send those performative-types, whereas receiving ability enabled means that the agent can accept that kind of performative-types and process.

```
Agent name: milling_machine_1
Agent-role: machine agent

request_to_load_task :
    input specifications: [job_id ]
    output specifications: [response-type, job-id]
    sending ability: enabled
    receiving ability: disabled

request_tool_loading :
    input specifications: [tool name]
    output specifications: [response-type, tool name]
    sending ability: enabled
    receiving ability: disabled

request_job_specifications :
    input specifications: [job_id]
    output specifications: [response-type, job-id, job specifications]
    sending ability: enabled
    receiving ability: disabled

.....
.....
```

Fig 5.2: An Agent file

5.4 Online and network Services

There could be some tasks that require heavy computation, which is not possible to perform on edge devices where the agent is installed due to the limited memory and computation power of edge devices. In such cases, services are utilized by the functions. Services are nothing but an internal function that lies on the server within the network or on the cloud where abundant computation and memory resources are available to run that internal function. These services can be called using APIs. The communication between service and function can be made secured with the help of credentials of function and encryption.

Services and function are a complement to each other and comes as a bundle. The developer of function also develops the service. Also, these services are only called by their partner function and not by any other function. So, there is no need to standardize communication between them.

An example of service use could be, let's say agent-1 wants to resequence its job queue as a new job added to its queue. Resequencing is a computationally heavy job, so we can create a service that resequences the queue based on all job's data. There could be a sequencing function on the agent that accepts the new job assigned to the machine and can utilize this service to resequence its job queue.

In the case of function which learns from live environment data, the learning process requires heavy computation for continuous weights update. So, learning process can be shifted to the cloud or local server in the form of service. The updated trained model can be downloaded from time to time to the agent to take local decision learned through data. In this case, even there is a loss of network connection agent can still make the decision best on the last downloaded model.

Chapter 6: Server of multi-agent system

6.1 Server network access

The server and agents should be connected in the same Network. This will enable the Agents to discover the host server within that network while trying to connect. The server has to be provided with HOST IP ADDRESS and PORT number on which this server will reside.

Currently in testing phase HOST IP is set to localhost and a free port is assigned to the server manually. The HOST IP address of the system on which the server is to be initiated can be obtained in various ways. One such way is to type out “IP config” in the power shell or command prompt. It will show the system IP address for the current network.

6.2 server initiation, handling new initiated agents

The code for server has the mention of HOST IP address and port number mentioned manually. A Web-sockets server can be initiated with these 2 inputs on a computer. Next there are 2 major functions that continuously run in the server. One of them is a function for accepting connections from newly initiated agents. When a new agent initiates it pings the server, and server registers this agent through this “receive” function. The second function is active on a new thread to handle messages from this Agent.

After the server is setup and running, it will continuously be sniffing for the initiated agents. When launching the agent, the agent has to be manually provided with the HOST SERVER IP address and PORT number corresponding to the server machine. The server establishes a TCP connection with this Agent and registers its IP address.

As a next step server asks the Agent for the AGENTNAME and agent directory, this name is appended in the Agents list and the directory is stored in the directories folder which may be updated. The agent directory is stored with the server with other agent directories at a particular path.

After this for each initiated Agent the server starts listening to that client on a different thread. Each Agent gets a different listening thread in the server, this server thread ends whenever the Agent goes out of operation.

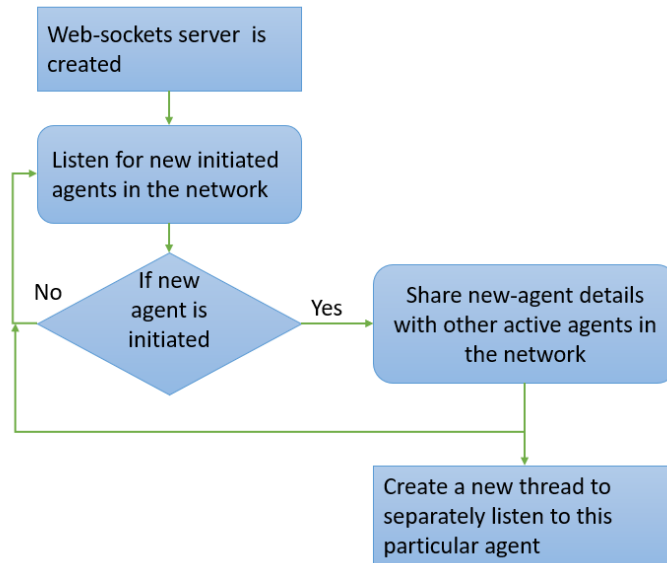


Fig 6.1: server initiation and handling new initiated agents

6.3 How server handles messages from the Agents

Server maintains a list of client objects and a list of the corresponding client names. When a client message is received by a server it is received on the particular thread on which the client messages are handled on the server. The message is first deserialized (i.e., converted back to JSON format) and first it is checked whether the message is a FIPA message or not. If it is a FIPA message, the same message is forwarded by the server to the receivers mentioned in that message. However, if it is not a FIPA message that has been received, where that agent had to talk to server itself, then those messages are handled separately by the server.

Chapter 7: Agent

There are two kinds of agents present in the system Machine Agent and Role Agent. They share almost similar characteristics with minor functionality changes. These are explained in subsequent points in the same chapter.

7.1 Machine Agent architecture

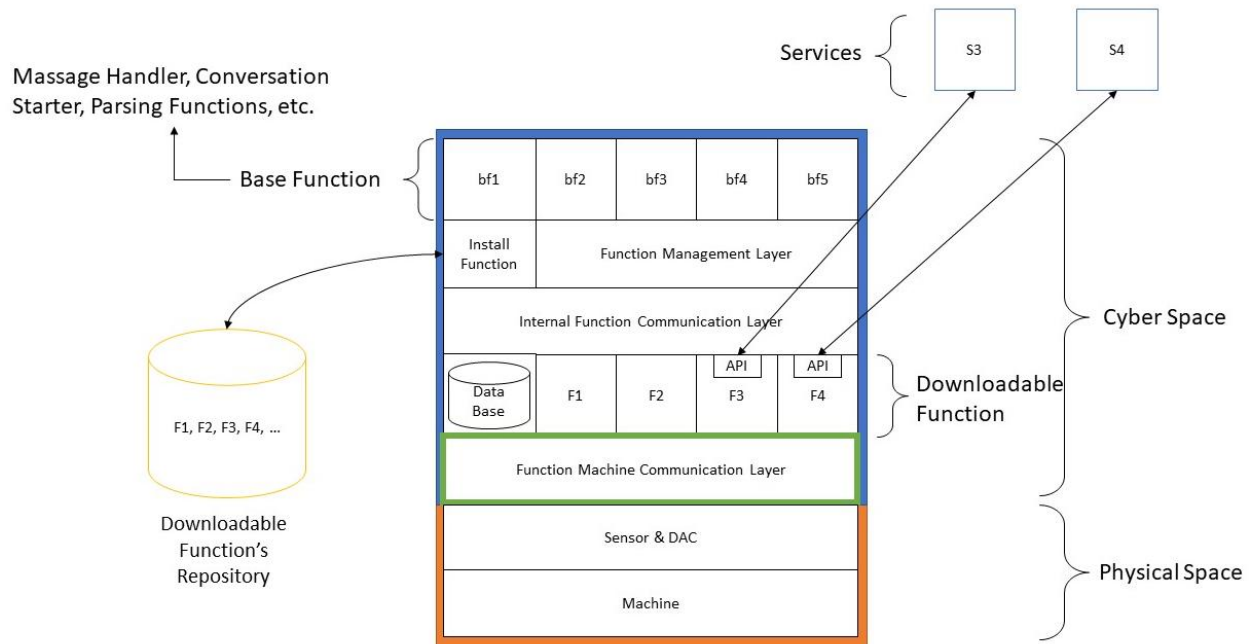


Fig 7.1: architecture for a machine agent

The figure shows the basic architecture of the machine agent. As shown in fig, the Machine Agent lies on top of the physical machine. It has capabilities to communicate with the physical world using sensor and actuators using function machine communication layer. The communication layer could be machine manufacturer-specific. The layer consists of a function that enables I/O to collect sensor data and pass command to the physical machine. The downloadable function can communicate to the physical machine using these I/Os.

All machine agent consists of basic functions denoted by “bf” which enables the agent to communicate with other agents. These consist of parsing, message handler, conversation starter, etc. The details of base functions are given in subsequent chapters.

Downloadable functions are the function which gives specific objectives to the agent. These are denoted by ‘F’ in the figure. There could be multiple functions that provide multiples objectives to the same machine agent. These functions do not come with a basic agent. These are needed to need downloaded from the function repository and installed. An example of such a downloaded function could be the sequencing function. Once downloaded, it enables the agent for optimal job sequencing. Another function could be the condition-based maintenance function which, if downloaded, enables the agent to predict

the remaining useful life of the machine. The third downloadable function could take data from both the previous function, i.e. sequencing function and condition-based maintenance function, to take more optimal sequencing decisions.

Some functions may have the corresponding service associated with them. These services are called by function using APIs. The developer of the function could also develop the associated service and provide both as a package. The services are nothing but some generic functions which require a lot of computation power or memory which is not available at the edge device where the agent is installed. These services could be run at the cloud or any other server within the network. The communication between function and service needs not to be standardized as they both are developed by the same developer.

The internal communication layer enables the data flow between different function. It is also a communication enabler between the base function and downloadable functions. This could be a separate function. But in the current version, it is symbolic and represented in the architecture. But in actual practice function directly call other function and communicate the required data in the standard data format.

Install Function block, as shown in the figure, is nothing but a separate program that helps to install the functions. These functions could be downloaded from some repository like 'GitHub' or any other platform. The detailed function installation is explained in subsequent chapters.

The function management layer is a representation of configuration files that are present in the agent. The user can change configure the agent and its function by editing these files. These configuration files help to decide which function to call in a specific scenario, which is the default function to call to achieve each objective; if the function requires communication with another function, to which function it should relay.

The database is nothing but a directory with large memory availability. This memory can be utilized by function to store their respective metadata in their respective data formats like CSV, SQL, TXT, etc.

7.2 Role Agent and its architecture

The overall architecture of role agent is similar to machine agent. But all the other agents in the multi-agent systems have a different role than that of a machine agent. These are assigned specific roles such as a jobs manager, material handler AGV or inventory manager agent etc. The Role agents require custom functionalities specially made for the demands and goals for which that agent has to be used.

Role agents have specific functionality on the shop floor. It will monitor all the events related to that specific function of the shop floor. In short, the role agent has only one objective, unlike machine agents, which can have multiple objectives. The installation procedure for role agent is similar to machine agent. After installation, the user has to download those specific role agent functions into the agent, which meant for a specific shop floor functionality.

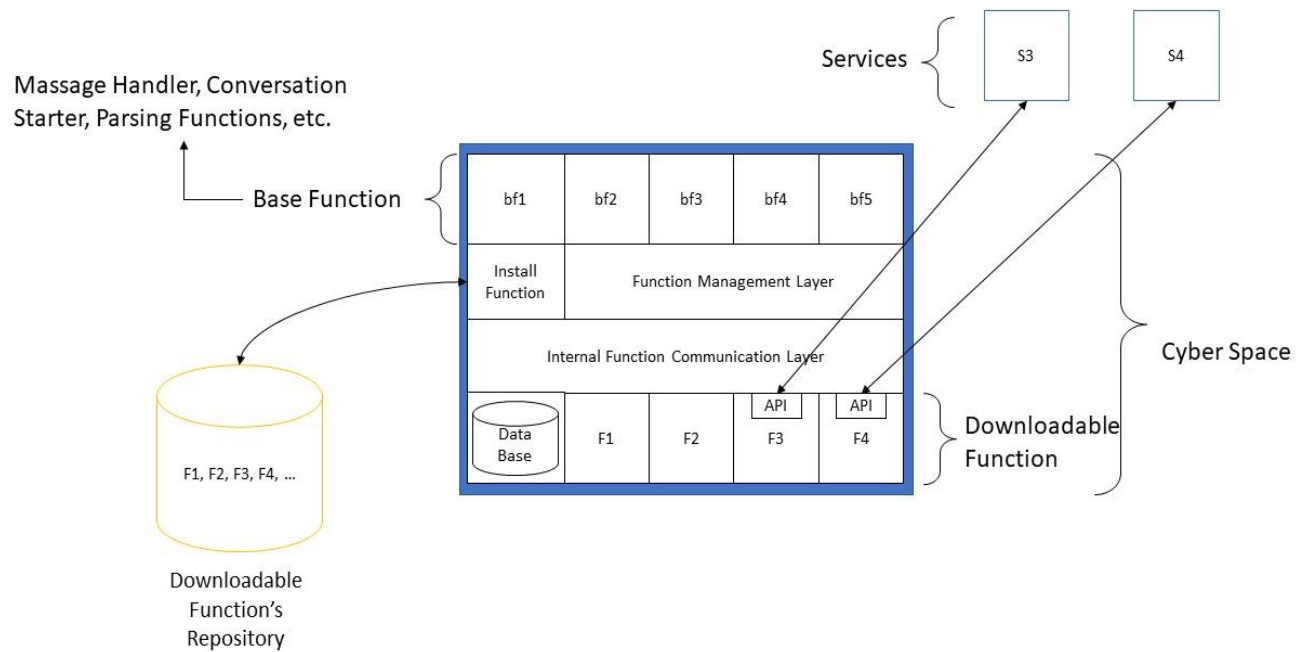


Fig 7.2: architecture for a role agent

Consider user has installed an agent, and now he/she want it to transform into a role agent specific to sequencing. Then the user will download sequencing related role agent functions only into the agent.

There are certain situations where users need to communicate to all the agents, like introducing the new batch of job on the shop floor or changing the due date of a certain batch in case of sequencing. There should be an agent that provides GUI to the user so as to pass information to all the agents. When the customer places an order, using which batches of jobs are planned. These planned batches need to be communicated to all the agents. The role agent plays this communication bridging role. It also acts as a blackboard to other agents for a specific task, i.e., it maintains a copy of metadata of all the agents related to that specific function to build redundancy in the system.

Types of role agent include material handle system, job manager agent, inventory control system and an operator HMI.

Example: Job manager agent (role agent)

The job manager agent accompanied with a GUI as shown below, is a special role agent that the operator can use to initiate a new job. This agent is responsible to have and interaction (mostly auction) with the other agents and assign the new job. This agent may not be associated with a physical machine, it just requires a computational hardware to reside on.

The screenshot displays the 'Job_Manager: manual FIPA messaging' interface. It includes a form for adding new jobs with fields for Job Id, processing Time, Due Date, Early Penalty, and Late Penalty. A table lists active agents: machine-Agent 3, machine-Agent 2, machine-Agent 1, and Job_Manager. A table at the bottom shows job details and allotments for five jobs, with the fourth job (sr.no 4) highlighted. A conversation log on the right shows the sequence of messages between the Job Manager and machine-Agent 1, including a call for proposal and an acceptance.

Job Details Form:

- Job Id:
- processing Time:
- Due Date:
- Early Penalty:
- Late Penalty:
- Buttons: add new job, Di change, Pi change, load selected job

Active Agents:

- machine-Agent 3
- machine-Agent 2
- machine-Agent 1
- Job_Manager

Job Details and Allotments Table:

sr. no	Ji	pi	di	ei	Li	allotment
0	2	23	06-25-2021	21	32	machine-Agent 1
1	3	26	06-29-2021	22	31	machine-Agent 3
2	4	28	06-29-2021	22	31	machine-Agent 1
3	13	1	08-19-2021	2	3	machine-Agent 2
4	7	24	06-26-2021	3	4	machine-Agent 1

Conversation Logs:

```

Server ---> all
Job Manager is now connected
-----
Job Manager ---> ['machine-Agent 1'] @ CNIP-5WjA6Fn
protocol: contract_net_interaction_protocol
performative: call_for_proposal
type: cfp_for_new_job_arrival
Content: [7, 24, '06-26-2021', 3, 4]
-----
machine-Agent 1 ---> Job_Manager @ CNIP-5WjA6Fn
protocol: contract_net_interaction_protocol
performative: propose
type: propose_penalty
Content: 2072.0
-----
Job Manager ---> machine-Agent 1 @ CNIP-5WjA6Fn
protocol: contract_net_interaction_protocol
performative: accept_proposal
type: accept_proposal
Content: None
-----
machine-Agent 1 ---> Job_Manager @ CNIP-5WjA6Fn
protocol: contract_net_interaction_protocol
performative: inform
type: inform-done
Content: None
-----

```

Fig 7.3 GUI for job manager agent

As seen on the GUI screen there is a section where new job details can be added. The blank sections are required to be filled and new job has to be added there. As soon as the operator clicks on “add new job” button, the job manager starts interactions with all the agents that can accept a new job. The manager starts a contract net interaction and sends a call for proposal to the agents. Later it waits for 5 seconds to receive all responses and after this wait, it does assessment of all the proposal costs that have been sent by the agents. As seen from the conversation-logs section on GUI, the recent job has been allotted to “machine agent 1”. At the end of allotment, the manager displays the new allotment of job on the screen along with its details.

The button “load selected job” is used to fill the details of a job that is manually selected by the operator. Later the Due date change and processing time change buttons will inform the allotted agent for respective changes made on the GUI screen.

7.3 Agent functions

Agent functions is a superset of all the functions the agent uses. This includes base functions, active and passive functions, and downloadable functions.

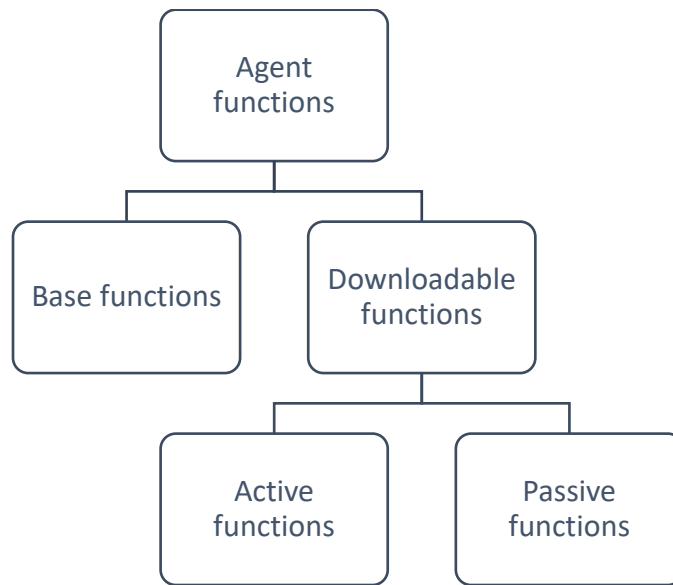


Fig 7.4: abbreviations of functions and their hierarchy

An agent function is nothing but a “.py” file containing different internal python functions. The internal python function means function defined by using the syntax: “*def FunctionName():*”. The Internal python file can be called by importing the function “.py” file in another file. It is necessary that the name of each function should be unique. We can categories the function by its activation, active or passive. Also, we can categories the function by its origin, i.e., base functions or downloadable functions.

7.3.1 Base Functions

Base functions are the functions that are essential for the agent to run in its very basic form. These base functions come along with agent installation. These functions are analogical to the OS of the PC/laptop, which is required to run other programs in the PC/laptop. The base function includes communication function and function related to invoking other functions based on communication needs. These do not include a function that provides an objective to the agent. An example of base functions would be a function that does message handling, FIPA message creation, serialization and deserialization of FIPA messages, defining receiving and sending functions of the agent, maintaining subscription topics, conversation control, etc.

7.3.2 Downloadable Functions

These functions do not come with agent’s installation. The user has to download them from some repository and install them in the agent. These functions provide an objective to the agent. As explained earlier, there could be multiple downloaded functions in the agent, providing multiple objectives to the function. An example of such a downloaded function could be the sequencing function. Once downloaded, it enables the agent for optimal job sequencing. Another function could be the condition-based maintenance function which, if downloaded, enables the agent to predict the remaining useful life of the

machine. The third downloadable function could take data from both the previous function, i.e. sequencing function and condition-based maintenance function, to take more optimal sequencing decisions. Some downloadable functions may have the corresponding service associated with them.

7.3.2 a) Active Functions

Active functions are those function which runs continuously doing some dedicated task. Whenever an agent is initiated, at the time of start each active function acquires a separate thread on the processor and runs continuously on that thread. The active function runs continuously till the agent is alive. If some error occurs causing stoppage of one of the active functions, the stoppage of one active function will not affect the other active function or any other function, and they keep running as it is. An example of active function could be condition-based maintenance function which continuously monitors the sensor data and predict remaining useful life of the machine.

Active functions cannot be accessed once they are initiated. Their work is only to identify triggers and start an interaction. They do not handle the replies to interactions. To handle such a situation, another class of function exist called passive functions.

7.3.2 b) Passive Functions

These are class of functions that respond only when called. These functions accept input in a particular format, and it produces output in a specific format. An example of such functions could be a sequencing function, which re-sequences the single machine queue only when asked for.

7.4 Agent startup or agent initiation

The code for Agent has the mention of HOST IP and PORT of the server. After fetching these credentials, the Agents establish a TCP connection with the server.

While starting a new agent, the server now asks the agent for its Agent-name and its agent file. The server publishes this information (agent name and agent file) to a topic named “new agent”.

There is a base function of Agent that is active on a thread and continuously looking to receive messages from the server. When the message is received it is further processed by other base libraries.

Agent initialization means running the main program file of the agent. Whenever an agent is initialized, the agent will first import all the function from Base Functions and Downloadable Function directories. Then it will import and read all the configuration files which are part of the function management layer. It will keep all the data in configuration files in the RAM of the system. The agent will start iterating over the active function from imported functions and start each of the active functions over a separate thread. Once all active functions are started successfully, it will connect to the server and handshake it. Now main program file will wait for triggers. If some trigger to call a passive function happens, it calls the passive function by starting that passive function over a separate thread.

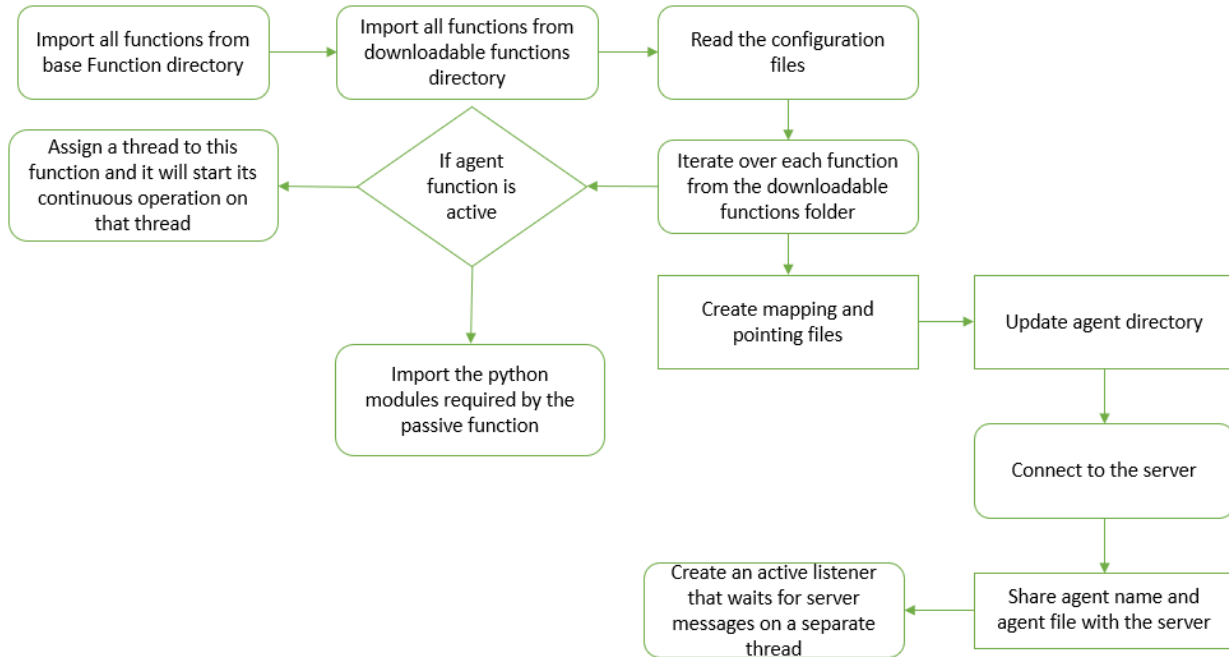


Fig 7.5: Tasks at agent initiation at the individual system level

7.5 Transfer of messages between agents

Sender Agent has a message object ready, so it is first converted to a JSON message and then this message is serialized or flattened. We make use of a python package named 'pickle', which is used for serialization and deserialization of python objects. So, the serialized message is now sent from the sender agent socket to the server socket via TCP protocol. When the server receives the complete serialized message, it deserializes the message back to JSON and extracts the list of receivers for this message. The server now sends the serialized message to all the Agents in the receivers list. Each receiver agent receives a serialized message. Now this message is deserialized to JSON message. And this JSON message is again converted to a message object for convenience of use. This completes the transfer of messages between agents.

7.6 Function classification

Each downloadable functions serves certain function. This function either tries to detect triggers or respond to messages posted by other agents. But there can be multiple number of functions that can serve the same purpose. For example, if an agent wants to calculate the RUL for machine, there can be different python scripts that it can access to generate the RUL value. Thus, all these functions will come under a common class. This class is called "function_type" in the introduction section of the function script. Each downloadable function has a function-type or class associated with it. Let's say condition-based maintenance (CBM) is a type of function. There could be multiple functions developed in the domain of CBM, so the type of all such function would be CBM. All functions of CBM type should accept input and produce output in a specified format which need to be standardized. Likewise, there could be multiple functions developed for single machine sequencing, and they all should accept input and produce output in the specified standard format. This leads to defining different function types and standardization of input and output for each type of function.

The standardization will help achieve inter-function communication as each function should know the data format acceptable by other functions. The acceptable format can be understood by knowing the function's type.

7.7 Major base libraries or base functions of an Agent

The base libraries have a defined path, where they will be put during installation of the Agent. Base libraries of an agent define how agents basically parse and send messages and the logic for response to messages.

The base functions are arranged in form of modules in a folder named as base libraries. The “messaging” module has the following functions:

1. flatten message: this would take a JSON file and serialize it to string via PICKLE.
2. unflatten message: This would take a serialized format and convert it back to the object it was before it got serialized, in this case the output will be a JSON file.
3. FIPA message class: Since the parameters of a FIPA message such as sender, receiver, performative, protocol, etc. are known, therefore generalizing it by creating a FIPA message class will help directly create a FIPA-message object. The objects can directly be created by referencing this class.
4. Message to JSON: this base function will take an input as a FIPA message object and convert it into a JSON file to give output.
5. JSON to message: It takes in the JSON message and converts it to a FIPA-message object.

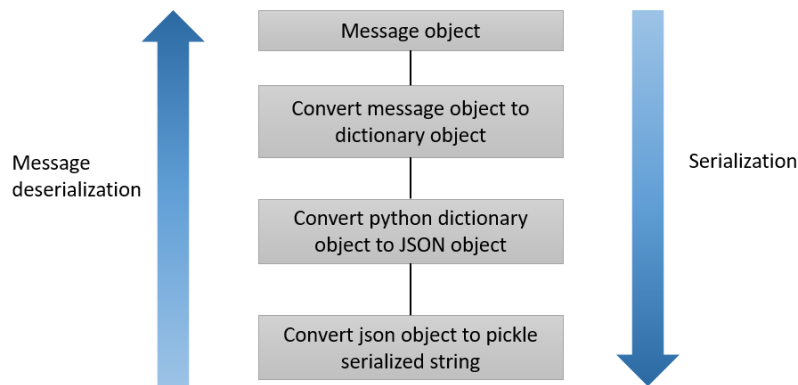


Fig 7.6: serialization and deserialization of a message

7.8 Message handling

When an agent receives a new message, then the message is handled by the parent message handler function. But before sending the message to parent message handler, first it is checked whether the message is FIPA message or dummy-FIPA message. Dummy-FIPA messages are used for conversations with the server only, such as topic publishes or asking for agent info etc. This check happens after JSON to message object conversion. Each function that is mentioned in the figure is explained below.

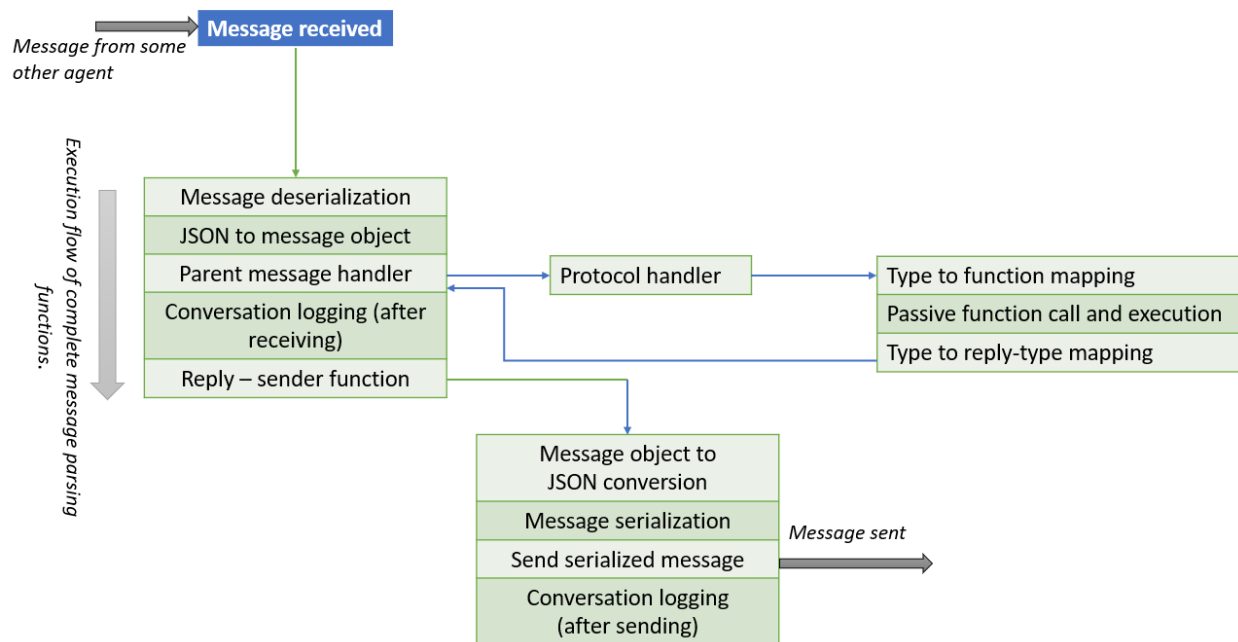


Fig 7.7: All message parsing functions

7.8.1 Parent message handler: This base function is called to parse the message object. Its inputs are the message object and the client itself. It performs the following actions:

1. The protocol of this message is identified and a protocol handler function is called.
2. the parent handler has to send the message after the reply has been generated through a reply sender function.

7.8.2 Protocol handler functions: any standard FIPA protocol thus identified by the parent message handler has a corresponding protocol handler. This protocol handler further identifies the performative of the message, sets the reply performative according to standard protocol flow and calls a type-to-function mapping function.

3. The mapping function: the mapping function does the following things:

- a. **Performative-type to function-type mapping:** It accesses the agent directory to map out the function for a specified performative type. This also sets the 'reply to type' parameter of the reply message to the current performative type.
- b. **Function-type to executable function mapping:** As function type refers to a particular class or category of function and there can be many functions to choose from for execution of a performative type, therefore there is a text file named as "pointing.txt" that has the mapping of function type to executable function mapping. This file is editable by the user, else the agent sets some function as default.

c. **Function execution:** The content for the reply message is generated through this function execution, it may even throw an error message, which can be handled by exceptions which will show up on the console part of agent GUI.

A call to the protocol handler will generate a reply message that is passed as an argument when calling the reply sender function.

Reply sender function: The work of this function is to calling a conversation logging function and passing the FIPA reply through it. Further it will serialize the message before sending it through the WebSocket client.

7.9 Trigger identifications

The trigger identifications are done via the active functions. But to keep flexibility of rules on which these triggers are identified there can be a text file that comes with a functions package that a particular active function will access.

7.10 Identification of target agents

The continuously running active functions are supposed to be sending messages at identification of communication triggers. The FIPA message to be sent is already coded into these functions. As these trigger functions become a part of Agent logic the Agent decides the receivers for these messages. These agents can be identified using the agent files of other active agents in the network.

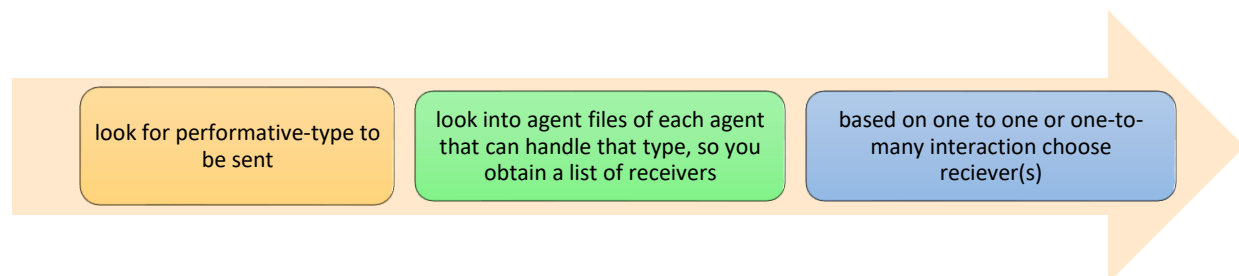


Fig 7.8: deciding target agents by target agent identifier

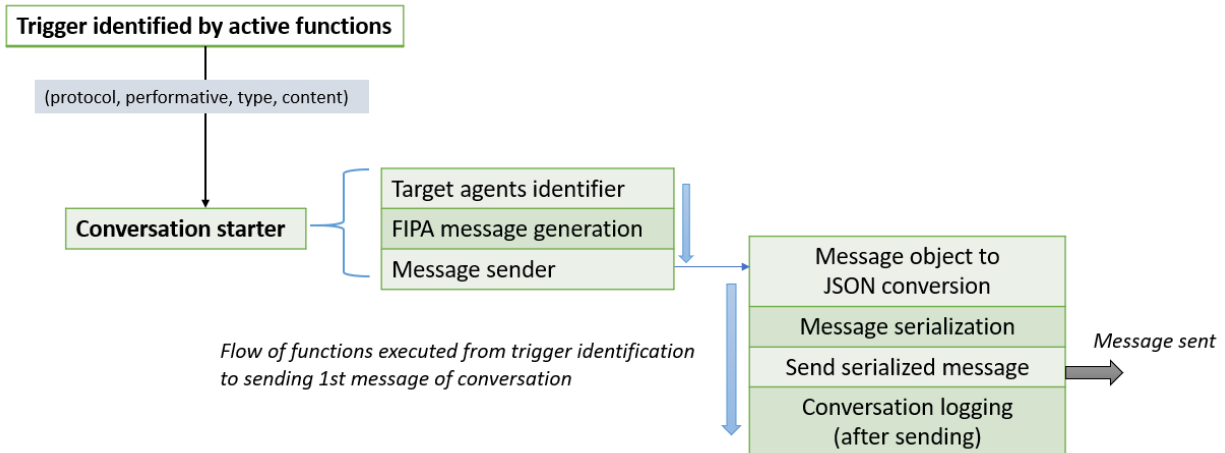


Fig 7.9: Conversation starter function and other base functions that it uses

When an active function, based on some logic decides to start a conversation and communicate, it would first call the conversation starter function.

7.11 Conversation starter function: This base function is called when the active functions identify a trigger, the active functions are supposed to pass performative, protocol, message type and message content to this trigger handler function. It has the following functionality.

1. Starting a FIPA message object and filling in the parameter i.e., protocol, performative, content and type.
2. **Target-agents identifier function:** based on the type, performative and protocol mentioned while calling the conversation starter function, the target agent identifier function will go through the agent description files of all the agents that can accept a particular performative-type. By this way the target agents are identified.
3. **Conversation logging:** The conversation logger function creates a new conversation-id and assigns it to the FIPA message.
4. After the above process the message is converted to JSON, serialized and sent via socket client.

7.12 Conversation logging (Log of all the interactions the agent makes with other agents)

There is a conversations folder that contains 2 subfolders, namely active conversations and ended conversations. Each of these folders have sub folders named after each protocol. In each protocol name folder, the conversation related to that particular protocol gets logged in form of JSON files. The conversation-id of the message is the most important variable here.

The way the conversations are logged are different. There are three major functions for logging and their tasks are as follows.

1. First function is to create a JSON log file. Mostly used by agent for logging the 1st message of conversation that is being sent.

2. Second function is to log and update a preexisting log of same conversation id. This means that a reply for which ongoing conversation is identified by the conversation-Id of the message, and the previous log is updated by adding this new message to it.
3. Third function is to update the pr-existing log and end the conversation. This means that agent understands that a conversation has ended and there would be no further message with this same conversation-id. In this case after logging in the final message of conversation into the log-file, this log file is removed from the current folder and pasted in the ended conversations folder

A conversation log basically is named after the conversation-id. For example, "RIP1" is the conversation -id for request interaction conversation, and its contents are mentioned below.

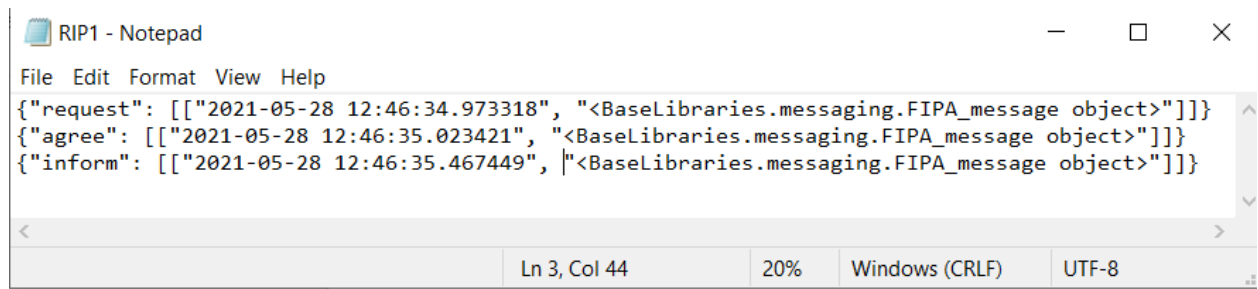


Fig 7.10: conversation log for a completed conversation that took place with request interaction protocol

The conversation logs help the Agent to understand which conversations are ongoing. This enables the agent to end any ongoing conversations, or search for previous conversations with other agents. If some agents are unresponsive the Agent can take a decision to end the conversation after some time.

Chapter 8: Integration of agent functions into the agent

8.1 how agents can be customized using downloadable functions

The functionalities of an agent come from various active and passive functions that it uses. These functions can be installed into the agents to enhance its functionalities. The active and passive functions govern how the agent behaves. The installation process of such functions is easy and is mentioned in function-installation section. These agent functions are stored in a folder named as “downloadable functions”.

An active or a passive function is basically a python module containing different functions inside it. An active or a passive function module should necessarily consist of two functions, namely “introduction function” and “execute function”.

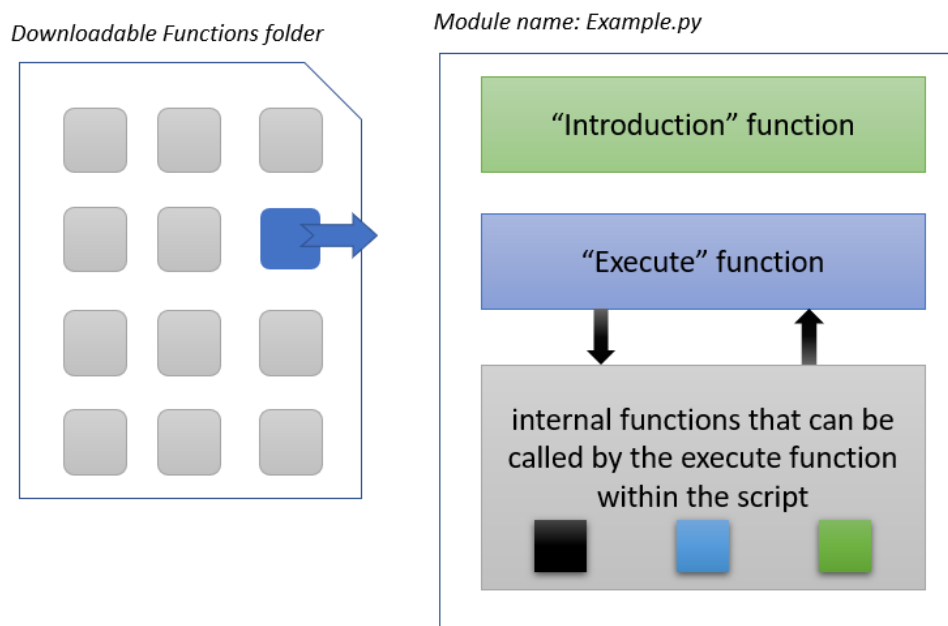


Fig 8.1: Structure of a downloadable function

8.2 Agent software files storage System

The whole agent is contained in the main directory called “Agent”; the user can rename this directory as per his/her choice. For explanation, we consider the directory name as “Agent”. The directory structure is shown in the figure.

The main directory contains “DownloadableFunctions” directory, which contains all the downloadable function downloaded and installed in the agent. This directory represents the Downloadable Function section of the agent architecture. The “BaseFunctions” directory contains all the base functions of the agent which comes with agent installation. This directory represents the Base Functions section of the agent architecture. All the files contained in the “DownloadableFunctions” and “BaseFunctions” directory are “.py” files.

The “Database” directory represents the Database section of the agent architecture. It is nothing but free space or a local drive where functions can save their data or metadata. Each function can save data

in its suitable format. So, folder can contain mixed format data files created by different functions. For example, function1 saves data in “Database” directory in “.txt” format while function2 save its metadata in the same directory in the “.csv” format.

The “main.py” file is the main program file of the agent. To initialize, the agent user needs to run this file. It holds the main control of the agent. The agent initialization process is explained in the subsequent chapters.

The “install_function_file_gui.py” along with “UpdateAvailableFunctions.py”, “UpdateFunctionDependency.py”, and “UpdateActivePassiveFunctionsLists.py” represent the Install Function block of the agent architecture. These files together are utilized to install a new function to the agent. The detailed installation process is explained in the subsequent chapters.

All the “.txt” files represent the Function Management Layer of the agent architecture. These files are used to configure the agents and manage the functions. These files help in the communication between agents. It tells agent’s main file and all the function which function to call for a certain situation. All these files are user-defined. They are explained in detail in subsequent chapters.

New function installation files UpdateAvailableFunctions.py updateFunctionDependency.py Function installation GUI	Agents Directory MachineAgent1.json MachineAgent2.json	Downloadable functions DF1.py DF2.py ...	Other important files agentfile.json Function_dependency.txt Main.py
Configuration files Function_pointing.txt Type_to_function_mapping.txt Ip_address.txt	Database Data used by agent in csv, SQL and txt formats	Base Libraries conversationLogging.py Handlers.py Messaging.py	Conversation logs Active conversations Ended conversations

Fig 8.2: Agent folder file system

8.3 generation of mapping and pointing files

mapping (performative-type to function-type mapping) basically means to select a category or a class of functions called as “function type” for the performative type received by the agent. And pointing (function management) means to point this “function-type” to an executable function out of a set of executable functions.

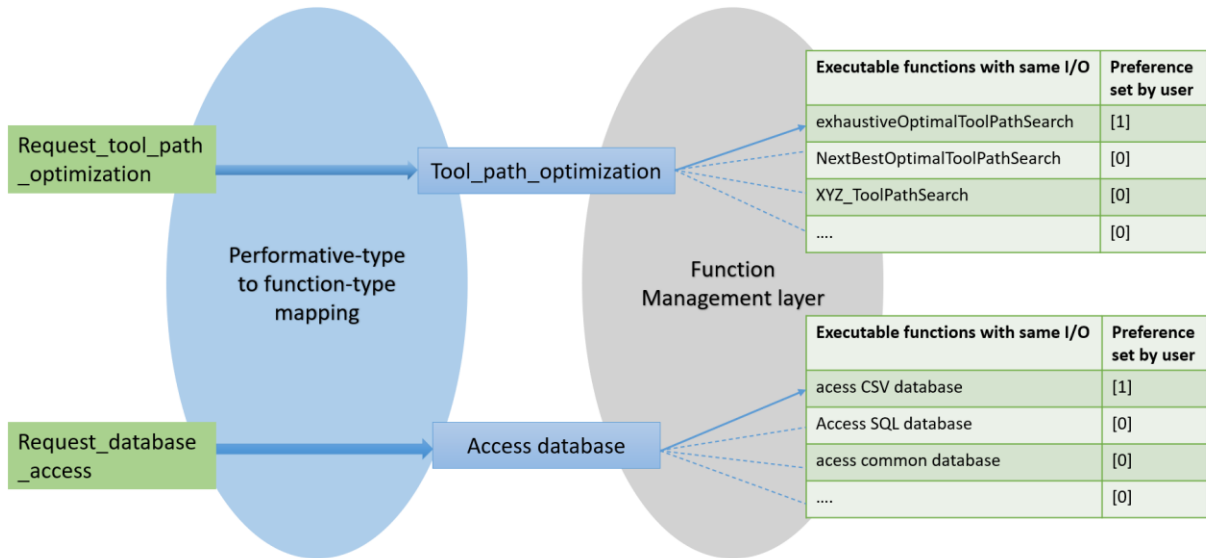


Fig 8.3: Mapping and pointing operation

When the Agent initiates, it first goes through the introduction part of all the downloadable functions. The Agent reads the introduction part of the function (function module i.e. “.py” file) and it knows which function is active or passive, and what function type it should be placed under as well as which performative this function-type can serve.

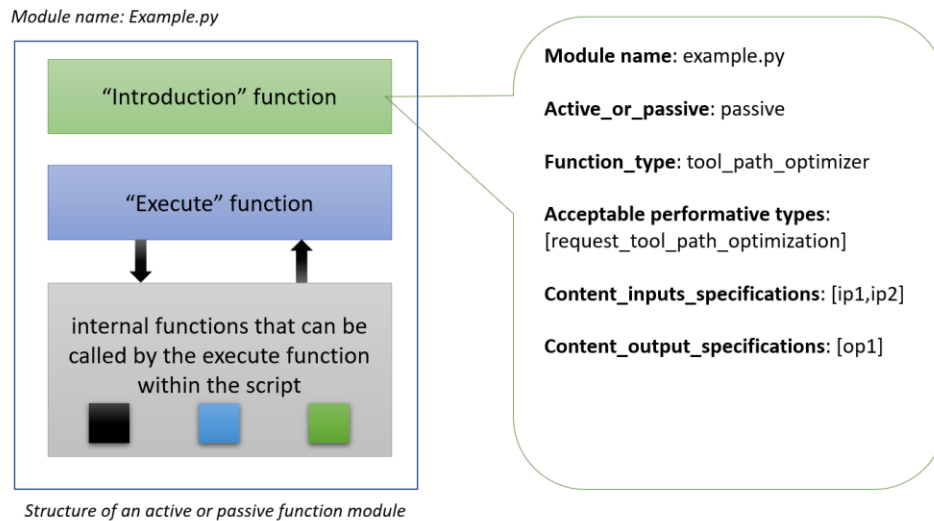
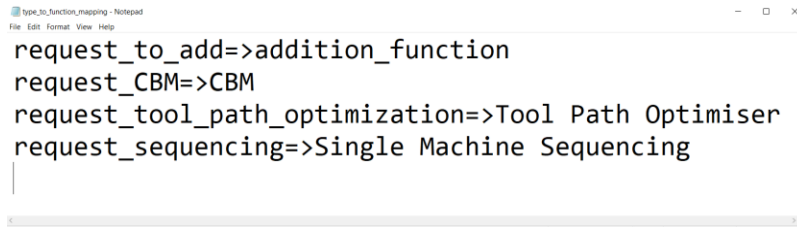


Fig 8.4: description of introduction function (internal) of a downloadable function

Reading the introduction part (internal function) of each function when agent starts, the agent first creates a mapping file in “.txt” format. In this file, the performative_type and the function_type or the function class is mapped using “=>” sign. As shown in figure below.



```

request_to_add=>addition_function
request_CBM=>CBM
request_tool_path_optimization=>Tool Path Optimiser
request_sequencing=>Single Machine Sequencing

```

Fig: function-type to function-mapping file

The import file for function management layer is the pointing configuration file. This file is created at start of agent and the agent maps function-type to names of various functions and their preferences as explained in function management section.

The agent basically calls for the “execute” function within the function module that finally gets selected according to preferences, further on the execute method may call for other internal functions or downloadable functions, which is not a concern of the agent. Only the execute method should generate an output as required for a particular performative-type.

8.4 Function Management Layer

There are different configuration files of the “.txt” format present in the Agent Folder. These files help in the smooth functioning of the agent. These all files combined called the function management layer of the agent architecture.

Utility of this layer is to decide which exact function has to be called for amongst the available functions for processing some tasks. It is majorly formed by some text files as explained below.

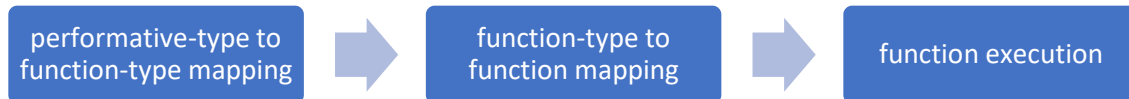
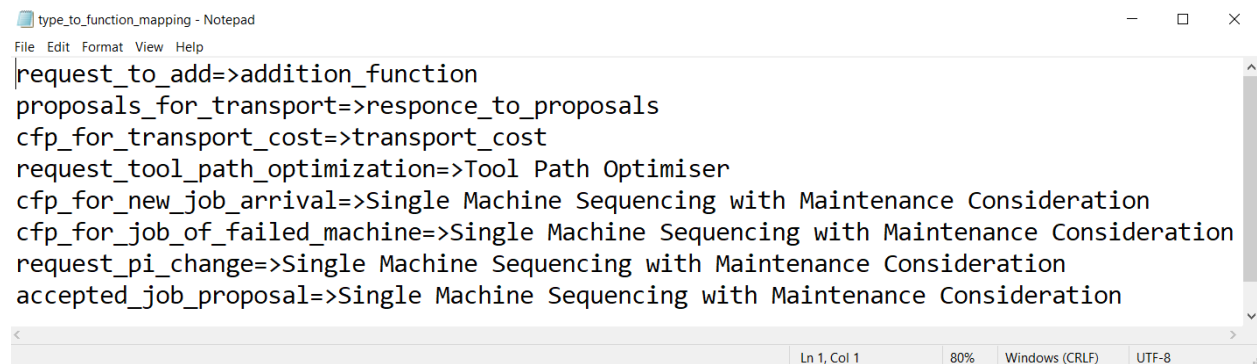


Fig 8.5: process of selecting an executable function

The “functions mapping.txt” file is a file generated at agent initiation (is updated if it already exists), which consists of a mapping of performative-type to the function-type (class of function). This mapping helps the agent to understand which particular class of functions can handle this particular performative type.

As shown in the figure below, the type to function mapping file has each line in the format of, (“performative type” => “function-type”)



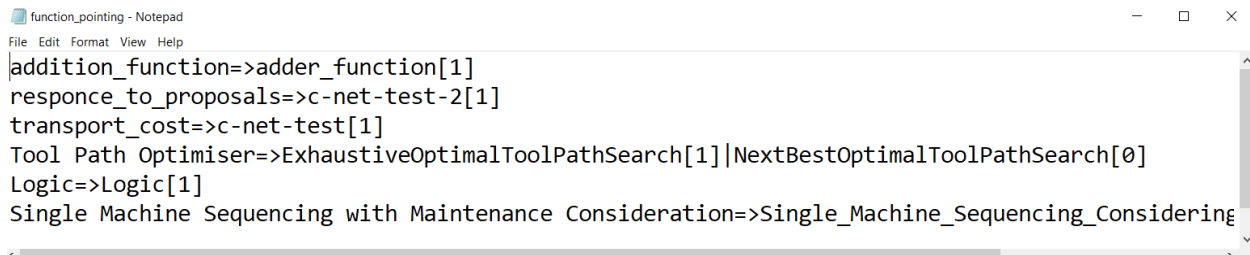
```

request_to_add=>addition_function
proposals_for_transport=>responce_to_proposals
cfp_for_transport_cost=>transport_cost
request_tool_path_optimization=>Tool Path Optimiser
cfp_for_new_job_arrival=>Single Machine Sequencing with Maintenance Consideration
cfp_for_job_of_failed_machine=>Single Machine Sequencing with Maintenance Consideration
request_pi_change=>Single Machine Sequencing with Maintenance Consideration
accepted_job_proposal=>Single Machine Sequencing with Maintenance Consideration

```

Fig 8.6: type to function mapping file

The “functionpointing.txt” is basically a configuration file that the user can edit. It basically comprises of all the functions inside a function-type or a function-class. As shown in the figure below, we can see that the function-type named as “tool path optimizer” has two executable functions separated by “|” character, that can be called for execution. The notations [1] is to be used for whichever function the user wants the agent to use, or else agent randomly uses a function in that class. The functions under each function class, have same standard input and same standard output.



```
function_pointing - Notepad
File Edit Format View Help
addition_function=>adder_function[1]
responce_to_proposals=>c-net-test-2[1]
transport_cost=>c-net-test[1]
Tool Path Optimiser=>ExhaustiveOptimalToolPathSearch[1]|NextBestOptimalToolPathSearch[0]
Logic=>Logic[1]
Single Machine Sequencing with Maintenance Consideration=>Single_Machine_Sequencing_Considering
```

Fig 8.7: function type to function pointing file

Chapter 9: system robustness

9.1 Hot standby server?

The server shutdown will not affect the agents, but it will affect the communications among the agents. A hot standby server is present in the system, which lively copies all the metadata present in the main server. Both the servers are running on different IP address and in different machines, so that failure of one should not affect the other. All the agents will have the IP address of both servers. These are provided while installing the agents. Whenever the heartbeat from the main server stops, agents come to know that the main server is failed, then immediately they start communicating via hot stand by agents. Also, as explained in the previous section standby server takes communication controls in its hand when it stops receiving the heartbeat signal from the main server. When the main server comes online again, it copies all the metadata from the hot standby server and broadcast the message to all the agents. Then all agents again start communication via the main server. Also, it starts sending the heartbeat signal to the standby signal so that the standby server comes to know that the main server is live again, and it starts acting as a standby server again.

9.2 Sudden machine agent stoppage

In this case, it will stop sending a heartbeat to the server. The server will identify the agent has failed, and it informs all other agents via broadcasting a message. In the working condition of all agents, keep an updated copy of data of specific functionality with the role agent. So, the role agent also acts as a blackboard for all agents, considering the particular functionality. Consider an example; machine agents are performing sequencing and condition-based maintenance (CBM) task. Whenever there is a change in the individual machine's job queue, it will send an updated queue to the role agent specifically meant for the sequencing related work. The sequencing role agent will have live updated respective machine queues for all the machine agents. While the role agent meant for CBM related work will have live updated CBM data for all the agents.

When a machine agent is revived after some time, it can copy all the previous data related to specific functionality from role agents related to that functionality. The agent can also view activities that happen in between and data of other agents of the manufacturing shop floor so that the machine agent becomes aware of the current condition of the shop floor and start regular working.

9.3 standby role agent

A hot standby can be maintained for the role agent. It will copy live data from the main role agent. When the main role agent stops sending the heartbeat signal to the server, it broadcasts to all the machine agents to communicate to the hot standby role agent.

When the main role agent is revived after some time, it can copy all the previous data related to specific functionality from standby role agents related to that functionality. It will start sending a heartbeat signal to the server, and the server will broadcast that the main server is live now.

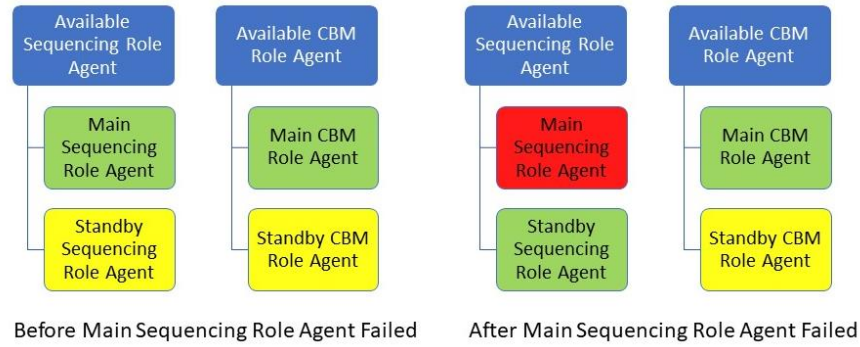


Fig 8.9: robustness against role agent failure

Consider the following example. All machine agents will have a list of available role agents for the specific task in the network. One of the available role agents is made default for the communication denoted by the green color in the diagram. Once the main role agent fails, it will broadcast its failure; then all the machine agents will make standby role agent as default role agent for future communication till the server broadcast the main role agent is live again. Once the main role agent is live again, all the machine agents will again make it the default role agent for communication.

Chapter 10 Previous approaches used and their drawbacks

There were previous approaches made to work with the agent-based system software available previously. Those approaches and their shortcomings are mentioned below.

10.1 PADE (python agent development platform)

The python agent development platform is open-source project under MIT license terms and is developed by Smart Grids Group (GREI) in Department of Electrical Engineering by Federal University of Ceará, Brazil. PADE is primarily built for the implementation of multi-agent systems on power systems.

For PADE a platform refers to server as well as the set of agents that are directly associated with this server, irrespective of the hardware these agents are on. For example, a server on machine-1, and 2 agents on machine-2 and machine- 3 respectively that are registered with this server, all come under a single platform.

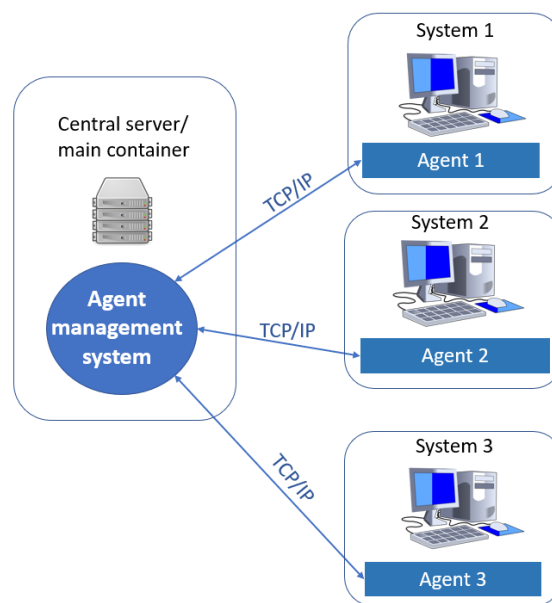


Fig: architecture for PADE agents

Each platform has a unique server which is named as agent management system or AMS. This AMS is responsible to register each agent that has been initiated on the platform, and acts as a middleman for transfer of agents from 1 agent to another. The messaging amongst these agents takes place with XML format and network protocol is TCP/IP based connections.

GitHub repository link for PADE: <https://github.com/grei-ufc/pade>

10.2 Drawbacks and challenges while implementation of PADE:

1.PADE is a software under development: PADE has a repository on GitHub. It is updated by a community of contributors and has an untimely update. In the current state of the software, it allows to initiate an AMS server, with agents on a single platform. These agents are able to interact with each other in FIPA messages.

2. Partial documentation: A major problem while implementation of PADE has been its partial documentation. There are some guidelines as to how to initiate agents on a single platform, and some examples as to how to use interaction protocols modules. But it lacks in explaining in how does this work for multiple platforms i.e., it is now well documented as to how the server recognizes agents that have been initiated on a different computer.

3. Slow agent responses in network: It was observed that PADE agents have very low response time to initiate and start running, this cannot be said when they are terminated though. Either it takes a lot of time for the server to recognize that the agent has terminated or the server never recognizes this transition of state of agent and freezes the agent activities on the platform. Therefore, it does not seem fit for scalable application.

4. Flask server and twisted library dependency: PADE agents and AMS are initiated in a container and utilize twisted libraries. It also utilizes Flask server for AMS, both of these libraries are older technologies and impose a dependency on the software, as these libraries are not readily available with default python packages and specific versions of them are required for the system to properly operate.

5. Partial FIPA compliance: PADE has only compliance to 3 interaction protocols i.e., request, subscribe and contract-net interaction protocols. Also, the message parameters do not include conversation-id which is important for conversation logging.

6. No conversation logging, and conversation control: The agents cannot identify different conversations with the same agent, as it does not make use of conversation ids. This specifically creates problems when the software is used at a large scale, when agent may have to handle a lot of messages at the same time.

While trying to implement PADE, the above reasons were identified. Further the FIPA compliance was increased to 5 interaction protocols and full message parameters were introduced during messaging, but further development on PADE was halted because there was no way to interact with the agents beyond a single machine as stated in above mentioned drawbacks.

10.3 Smart python agent development environment (SPADE)

SPADE is another agent development environment based on XMPP network protocol, open source under MIT license and coded in python. It has an XMPP server either online public server or an XMPP server in the network of the agents.

Just like PADE, SPADE also has a similar client-server network architecture, just the network protocol is XMPP. SPADE uses “asyncio” a default python library that is used for asynchronous communications.

A jabber-id is to be created as an agent-id which will be used by all the agents to identify the target agents. This id has to be created on online publicly available XMPP servers, which are generally used for chat applications. An XMPP application software can also be used as a server application within a network that can provide jabber-ids.

SPADE GitHub repository: <https://github.com/javipalanca/spade>

10.4 Drawbacks and challenges while implementation of SPADE

1. XMPP network protocol and server dependability: The biggest problem associated with the SPADE software is that it utilizes an online public XMPP server. This has 3 disadvantages, firstly these public servers cannot be relied on for crucial industry applications as they might go out of operation suddenly, secondly even if we set up the XMPP server on the local network, the server comes as a robust application which cannot be configured to specifically work for agent-based applications, such as sharing and storing agent files, and lastly this additional XMPP server application is an added dependency for the system.

2. Application Under development: Similar to PADE, SPADE is also developed by a community of developers on GitHub. Till the recent development, it allows for creating an agent with a working Jabber-id. These agents connect to online public server and the agent messages are transferred via this online platform. There has to have a lot of development on the agent side such as agent GUI, conversation controls and full FIPA compliance.,

3. Partial FIPA compliance: FIPA compliance of SPADE is only in terms of performatives, there is zero compliance to interaction protocols.

4. No conversation logging, and conversation control: The agents cannot identify different conversations with the same agent, as it does not make use of conversation ids. This specifically creates problems when the software is use at a large scale, when agent may have to handle a lot of messages at the same time.

Chapter 11: Conclusion

A smart machine has numerous characteristics that make it into an autonomously functioning machine that can make decentralized decisions. However, it is important to consider a certain set of properties or characteristics that can be held as a standard for determining which machine is ready for industry 4.0 applications. ZVEI standards that have been considered in this project work, turn out to be effective in determining fixed and open set of guidelines to judge a smart product. Major emphasis of this piece of work has been kept on incorporating the property of communication amongst the smart machines.

JSON format as a messaging format is a well-known, lightweight and prominently used data exchange format, and therefore it was decided to use serialized JSON object strings for messaging amongst the agents. The FIPA-ACL works as a language of interactions amongst the agents and not merely a mode of data transfer amongst machines.

While implementation of the proposed architecture, a client-server network had been created with the use of web-sockets. Each Agent is a socket in the network and the messages between agents go through via the server.

11.1 What has been achieved on the server side:

- Initial protocols (at server side) for agent-server interactions during agent initiation in a network is defined.
- Mediating one to one and one to many interactions.
- Broadcasting to all agents as well as forwarding message to particular target agents are available functionalities.
- Identifying discrete FIPA messages and responding to certain non-FIPA messages as topics. Some of the messages are for interactions between server and agents and not agent to agent.
- If an agent comes inside a network or goes out of the network, in both the situations all the agents are informed of this activity.
- Stable operation of server even when some agent in the network fails
- Handling multiple agent messages at the same time.

11.2 What has been achieved at the Agent side:

- Initial convention (at agent side) for agent-server interactions during agent initiation in a network is defined.
- Agent file is created (override on previous) whenever the agent is initiated or updated.
- Type-to-function mapping files and functions pointing file is updated at agent initiation.
- Full interaction capability for request interaction and contract-net interaction protocols established. Templates of all other interaction protocols are ready for use.
- Starting of all active agents on a separate thread.
- Agent can make Initial message handling decision i.e. whether to utilize a function for processing message content or whether the agent itself can process the message (like non-FIPA dummy messages).
- Agent can wait for message replies from the other agents for timed interactions, like for contract-net interaction the agent must wait for all the proposals for certain period of time to gather replies from all the agents while not hampering its other operations.
- agent has a GUI that shows interactions conversation logs with other agents.

- Agent can recognize and distinguish one conversation from other conversation by the help of conversation ids and thus it is able to carry out multiple conversations with many agents at the same time.
- There is a provision of logging in all the conversations in the conversation logs folder, and at the end of the conversation the conversation file is shifted to the ended conversations folder.

GitHub link for project repository: <https://github.com/IITB-MAS>

Chapter 12 Future scope

The multi-agent system proposed and explained in this piece of work is an inception towards achieving a fully autonomous agents-based system. The further work that can be proposed for the agent-based system can include the following.

12.1 On the server side

- The server, acts as a mediator for communications between agents, it is therefore important to take measures to make the server Fail proof which may be achieved by having some additional backup power hardware. Also, there can be a backup server that the agents may contact when the main server goes down because of some reason, which occasionally copies all the data that the main server has.
- The server can have a centralized agent updating mechanism through which the agents can get updated. This way all the agents can add a particular functionality at the same time. There could also be a provision of updating only a particular set of target agents.

12.2 On the agent side

- As additional functionalities get added to the system, hence there would be a need of an Agent GUI for the operators, this may include showcase of conversation logs, button for voice input from the operator, buttons for agent restart, agent update etc.
- The role agents such as material handler and inventory control agents need to be custom built agents and therefor the coding for those agents is to be done on the basis of specifically decided set of tasks that these role agents will be needed to perform.
- The templates for the interaction protocols are ready, but for some interaction protocols such as FIPA broking and recruiting interaction protocol, specific cases are to be identified where there is a potential to use these protocols.

12.3 Installable application

The agent-based system can come as an installable application. The application comes as a setup file that can be downloaded. The downloadable files will be different for different OS of the computer. There is a pop-up window that appears when the setup.exe file is run, there would be a license agreement that the user has to agree on, as a next step the entire application is installed in the system. There is a notice of setup completion, and if user clicks finish, consequently other setup windows for dependable programs will pop up. For server as well as Agent there are no different applications. The same application can be used to initiate a server or an agent on the system.

12.4 Agent management

In the agent management section, there will be a “update Agent software” option. Clicking on this option the application will check for new version updates online. If the version is up to date, it will notify that the latest version is present or else there is a new window that pops-up saying that a new version of software is available and there is an option of “download and install update” or cancel, and the application is closed. There is a new window that pops-up showing “download and install update”. During installation the agent cannot be accessed by other agents. The software update will not affect the working of the previously

installed function packages. After this process is finished, the application can be started and an agent or server can be initiated.

12.5 Installing new function package to the Agent based system (decentralized)

There is a section of package management under the section of agent management in this application, here we can see the already installed packages. There is an option to add packages. This should open a pop-up window asking for “upload new package script”. There will be an option of “get script online”, by clicking on this option the user is taken to an online platform through the browser, the browser will open a webpage that has these package resources and their information. The function packages will be as a list of packages. Clicking on one of the packages will expand the package downward and there are some more square boxes to tick. Functions packages come with option:

1. Install the active function + passive functions
2. Only install passive functions

The user has to select the appropriate packages from the list and at the end there will be a “generate script” option. This script will be downloaded. Back in the agent package management section, this script has to be uploaded in the pop-up window and then click on “download and install new packages”. the application is closed and installation progress is shown on screen. After the installation, the application restarts and the agent start its function as usual.

Another way of installing these packages is through accessing the scripts that have been shared by the server through the “shared by server” section. This section actually accesses a defined path where all the package scripts that are shared by the server are stored and can be accessed.

References:

- [1] A. Zeid, S. Sundaram, M. Moghaddam, S. Kamarthi, and T. Marion, "Interoperability in smart manufacturing: Research challenges," *Machines*, vol. 7, no. 2, pp. 1–17, 2019, doi: 10.3390/machines7020021.
- [2] Acatech, "Implementation Strategy Industrie 4.0 - results," no. January, p. 104, 2016.
- [3] EFFRA, "Factories 4.0 and Beyond: Recommendations for the work programme 18-19-20 of the FoF PPP under Horizon 2020," p. 67, 2016, [Online]. Available: <http://www.effra.eu/factories-future-roadmap>.
- [4] N. Boulila, "Cyber-Physical Systems and Industry 4 .0 : Properties , Structure , Communication , and Behavior," *Tech. report, Siemens Corp. Technol.*, no. April, 2019, doi: 10.13140/RG.2.2.27890.76485.
- [5] Y. Liu, Y. Peng, B. Wang, S. Yao, Z. Liu, and A. Concept, "Review on Cyber-physical Systems," *IEEE/CAA J. Autom. Sin.*, vol. 4, no. 1, pp. 27–40, 2017, doi: 10.1109/JAS.2017.7510349.
- [6] B. R. Beudert, L. Juergensen, and J. Weiland, "Understanding Smart Machines : How They Will Shape the Future," *Schneider Electr.*, pp. 1–13, 2015.
- [7] S. Mittal, M. A. Khan, D. Romero, and T. Wuest, "Smart manufacturing: Characteristics, technologies and enabling factors," *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.*, vol. 233, no. 5, pp. 1342–1361, 2019, doi: 10.1177/0954405417736547.
- [8] Bundesministerium für Wirtschaft und Energie (BMWi), "Which criteria do Industrie 4 . 0 products need to fulfil ?," *Platf. Ind. 4.0*, p. 32, 2019, [Online]. Available: <https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/criteria-industrie-40-products.html>.
- [9] M. Kasunic, "Measuring Systems Interoperability: Challenges and opportunities.," *Carnegie-Mellon Univ Pittsburgh Pa Softw. Eng. Inst*, 2001.
- [10] N. Velasquez Villagran, E. Estevez, P. Pesado, and J. De Juanes Marquez, "Standardization: A key factor of industry 4.0," *2019 6th Int. Conf. eDemocracy eGovernment, ICEDEG 2019*, pp. 350–354, 2019, doi: 10.1109/ICEDEG.2019.8734339.
- [11] R. Evans *et al.*, "MESSAGE: Methodology for engineering systems of software agents," *Eurescom, Edin*, no. September 2001, pp. 223–907, 2001.
- [12] K. A. Group, T. Finin, D. Mckay, and R. Fritzson, *An Overview of KQML: A Knowledge Query and Manipulation Language*, no. July. 1992.
- [13] S. Vaidya, P. Ambad, and S. Bhosle, "Industry 4.0 - A Glimpse," *Procedia Manuf.*, vol. 20, pp. 233–238, 2018, doi: 10.1016/j.promfg.2018.02.034.
- [14] FIPA, "Index @ Www.Fipa.Org." [Online]. Available: <http://www.fipa.org/>.
- [15] Foundation for Intelligent Physical Agent (FIPA), "FIPA Request Iteration Protocol Specification," 2002.

- [16] (Foundation for Intelligent Physical Agents) FIPA, "FIPA Contract Net Interaction Protocol Specification," *Architecture*, no. SC00029H, p. 9, 2002, [Online]. Available: <http://www.mit.bme.hu/projects/intcom99/9106vimm/fipa/XC00029E.pdf>.
- [17] F. For and I. Physical, "FIPA Subscribe Interaction Protocol Specification," *Architecture*, 2002.
- [18] Foundation for intelligent physical agents, "The foundation for intelligent physical agents," *IEEE Comput. Socceity*, p. 1, 2013.
- [19] FIPA, "FIPA Propose Interaction Protocol Specification," *IEEE Comput. Socceity*, p. 1, 2013.