Multi-agent system for smart manufacturing

A project by

Saurabh Mandaokar

(Roll no: 193100081, 2021 batch)

saurabhmandaokar24@gmail.com

Aniket Shashikant Adsule

(Roll no: 183106003, 2021 batch)

aniket.s.adsule@gmail.com

Supervisors:

Prof Makarand S Kulkarni

Prof Asim Tewari

MECHANICAL ENGINEERING DEPARTMENT

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

2020-2021

Abstract

The manufacturing industry consists of different types of equipment with various makes following various protocols. Some of them are cable of communication, while others are legacy equipment with no communication capabilities. ERP software and equipment may follow some proprietary protocols or format for communication which leads to a lack of interoperability between them, causing ineffective utilization of resources and inability to respond quickly to changing customer requirements and unplanned events. To respond quickly, decentralized and autonomous decision making is required. This can be achieved using agents and digital twins having communication capabilities in the smart network.

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.

The case study is presented, which showcased the single machine scheduling function and service which helps all the machines to optimize their batch queue dynamically and in a decentralized way according to changing shop-floor conditions. There are multiple functions developed with different objectives like tool health monitoring, machine health monitoring, and single machine sequencing. The different developer may develop all these functions. In contrast, the agent is developed by a different developer. However, all these functions can be downloaded and installed into the agent and work together to cater to the larger objectives. The digital twin can be converted as a function that accepts real-time sensor data to predict the remaining useful life (RUL) of the machine and prescribes optimal maintenance action based on RUL is showcased.

In another case study, functions related to tool path optimization is showcased. The case study also focuses on the scenario where multiple developers develop functions of the same type, and these are simulations installed in the agent. If the agent has to utilize one of the functions among all installed ones, the configuration files help in the agent decision.

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, digital twins.

Table of contents

List of Tables

List of figures

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 improvement in 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 pieces of literature. Common terminologies such as “Cyber-Physical Systems” (CPS) and “industry 4.0 products” have similar descriptions 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 with 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 works of literature, the interpretation of smart machines by the authors is stated by the properties or characteristics of the smart machine that they consider. The subsequent section mentions the desired properties of these smart machines.

1.3 Properties of a smart machine

The following properties of smart machines have a mention in various works of literature. 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
21	Status	F, A

22	I4.0 compliant services	A
23	Virtual description	A, B
24	semantics	A

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

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 machines should be uniquely identifiable that signifies their digital presence, thus providing 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 heterogeneous 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 a system to reconfigure services and behavior due to changes in the 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.**Cybersecurity**: 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 the loop**: Using human expertise and knowledge for validation of certain tasks. Providing a 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 a 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 smart machines. ZVEI which is a major manufacturing association in Germany formulated the criteria for identifying a product as an industry 4.0 ready product.

These criteria help providers in the market to decide which products can already be labeled 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).

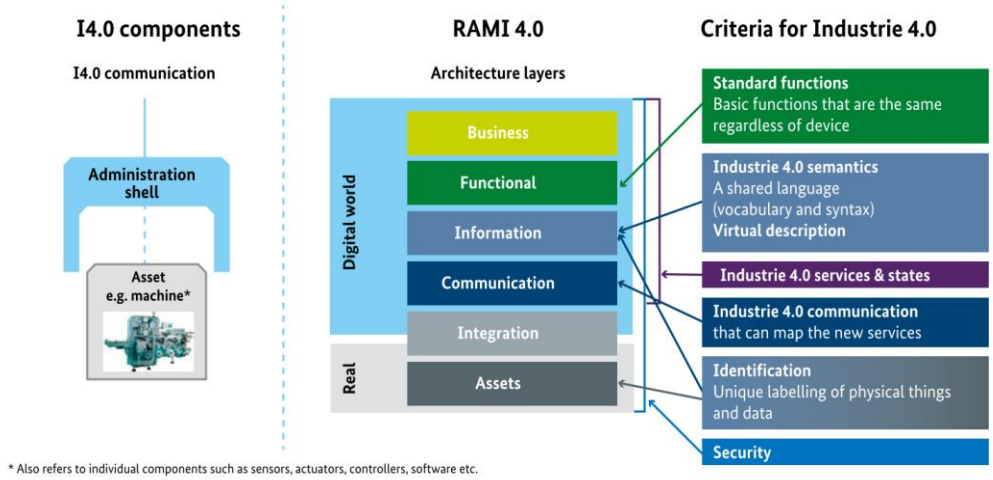


Figure 1.1: deriving criteria for industry 4.0 product from RAMI 4.0 architecture[8]

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 outrunner
Manufacturer independent identification of the asset with a unique identifier (ID) attached to the product, electronically readable. Identification in: **1)** Development **2)** Supply chain (logistics), production **3)** Sales, service, marketing **4)** Network	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 the 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
Standardized data in the form of attributes with cross-manufacturer unique identification and syntax for: 1) Commercial data 2) Catalog data 3) Technical data: mechanics, electronics, functionality, location, performance 4) Dynamic data 5) Data on the lifecycle of the product instance	T	M	For 2) catalog 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	Yes, via QR code
	T	M	For 2) and 5) catalog 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.	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. Virtual representation for the complete life cycle. Important properties of the physical component information regarding the relation between Properties relations relevant for production and production process-relevant relationships between Industry 4.0 components a formal description of relevant functions of the actual component and its processes	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, catalog, image, technical features, datasheet, etc. are available online
	I	M	Digital contact to service and 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

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
The 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 the OPC-UA information model following 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 the same data in the same functions. These basic functions serve as a base for the functionality, on which every manufacturer can build their extensions.	T	N	Functions described in Administration shell in the format of I4.0 sub-models.	First diagnosis and condition monitoring functions
	I	N	Functions implemented in Administration shell in the 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	A threat analysis has been executed. Appropriate security features were considered and publicly documented. Security by design. Security capabilities are described at the respective level (Authentication of the identifiers, user and role management, secure communication, logging of the security-relevant changes).	Security by design. Security capabilities are described at the respective level (Authentication of the identifiers, user and role management, secure communication, logging of the security-relevant changes).
	I	M	Available security capabilities are documented. Appropriate secure identities are available.	Is discussed and documented in the customer projects
			Security by design. Security capabilities are described at the respective level (Authentication of the identifiers, user and role management, secure communication, logging of the security-relevant changes).	

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
Transfer of product data and data files for interpretation or simulation, for example; product data in a standardized form.	T	M	The manufacturer makes data available/accessible online. The data should be relevant to customers and available/accessible with the assistance of identification/e. g. pdf via HTTP(s) and URI	Step Files, CAD drawings, etc.
The product can be addressed via the network, supplies and accepts data, Plug & Produce via I4.0- compliant services	I		The 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 can provide solutions at each layer of the automation stack. This means that the same manufacturer will provide for equipment, PLCs, 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 the 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 a 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 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 fewer repairs)
	Control on energy consumption (a service that monitors energy consumption)	Costs related to energy usage are optimized.

Retrieving Information from a part *in the development stage* by a service. (i.e., when the 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:** A variety of products can be produced when a separate production plan is available for each product.
MHS asks for the optimal path of travel to a service. Ex. if an MHS is instructed to go from the packaging section to the 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 Real-time track of all the moving entities on the shop floor can provide for the 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 that 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 availability data which it will ask using the M2M communications(service).	**Very quick data transfer and decision-making.** (in an industry with a considerably high number of machines, a lot of time will be wasted in finding out the availability of the required set of machines and further communicating this information manually, but in the I4.0 scenario this time is comparatively negligible)	**Better resource utilization** Job and Machine idle time reduces as a consequence of a reduction in time required for sequencing and scheduling

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 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
The machine tells the operator that it requires maintenance or repair. For example, there is a sudden blockage in the 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, the machine can identify the area of failure)	Machine downtimes during repairs will reduce.
Retrieving machine health parameters for maintenance by an expert. Ex. The expert operator wants to monitor the vibrations of the tool during a drilling process.	Tool degradation prognostics: (factors affecting excessive tool degradation can be monitored)	1. The reduced stock of spare parts. (tools) 2. The increased service life of parts. 3. Reduction in machine failures.

Table1.9: Communication between machine and operator

Reason to communicate	outcomes	benefits
Production of a batch is finished and it is to be sent further. Ex: Machine tells the MHS to carry the finished pallet to the next machine.	Shortest path selection: MHS decides the shortest path of travel when it is employed with the help of some service that manages shop floor traffic.	**Lowering of material handling cost.**
Machine broadcasts for knowing the availability of an MHS for transport of a pallet. Best-suited MHS is selected based on its capacity, availability, and presence in the vicinity.	Time-saving in MHS finding: Ex: in the 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.	**Optimization of material transport cost.** (based on bidding the best MHS will be selected for transporting the material based on cost optimization)

Table 1.10: Communication between machine and MHS

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

Table 1.11: Communication between machine and inventory storage system

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

Table 1.12: Communication between machine and job

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

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 the manufacturer which plays an important role in scalability and industry-wide integration of systems.

Chapter 2: Literature review

2.1 What is industry 4.0

The idea of industry 4.0 has its manifestation in various works of literature with different names. Smart manufacturing, Factories of the future, and connected industry are some of the common terms that are used in a similar context to that of Industry 4.0. Various manufacturing associations and works of literature have tried to define 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 pillars of industry 4.0

the following can be seen as pillars of Industry 4.0:

2.2.1 Big Data and Analysis

In Industry 4.0 scenario, data will be generated by a number of equipment simultaneously, here the volume of data, its variety, frequency of generation and its value is high. This leads to a need for big data analysis and sophisticated Machine Learning algorithms for real-time decision making. Recorded data can be used to find out the occurrence of any life event of equipment, any new issue or prediction of future events. Three main types of analytics can be used Descriptive, Predictive and Prescriptive Analytics.[9]

2.2.2 Autonomous Robots

Robots are becoming more intelligent, autonomous, flexible and cooperative nowadays. These can interact with each other and surrounding humans to work in a colligative way.[9]

2.2.3 Digital Twin

Simulation is now extensively used with the help of a digital twin. The mirror image of a physical entity is created in cyberspace, which can be a physical, data or combined model of a real entity. This model behaves the same as the real-world entity. Real-time data is collected from a real entity and fed to this model to simulate future condition. This can help in finding the current state as well as in predicting the future states of a given entity. A more detailed discussion is presented in the following document. [9]

2.2.4 Cyber-Physical Systems

Cyber-Physical System is a physical embedded device with software and computing power. It has self-management capabilities. In Industry 4.0, manufacturing equipment is transformed into a Cyber-Physical Production System, which is software enhanced machinery with computation power capable of taking decentralized autonomous decisions. It virtually replicates the state of physical machinery by using sensors.[9] It may also be facilitated with actuators so that it can make changes in its environment whenever required. CPS should be taken in the sense of a disconnected entity (with another CPS). CPS can have an open-loop or close loop control system. If it is an open-loop system, it only communicates the state of the machine and all sensor data in a forward way to some database. It may be a close loop system as well, which also receive data from other CPS or from the database so that it can take necessary decision and actions.

2.2.5 Internet of Things

This proved interconnectivity between objects that communicate in information using standard protocols. Using this IoT architecture, each object can communicate the real-time outcome of its interaction with the environment, current state, its own information, etc. Due to the availability of real-time data, one can take a real-time decision. This makes the system more intelligent and agile. IoT architecture can be utilized by humans in the loop, digital twins, smart sensors, production services, smart products, etc. [9]

2.2.6 Horizontal and Vertical System Integration

In the Industry 4.0 paradigm, it is essential that each entity should communicate with another entity. This can lead to data sharing between each entity. Data sharing can happen between different products, cloud

services, enterprise management system, third party agents, etc. As a demonstration of this, a multi-agent system with blackboard architecture is shown in figure 4. Cyber-Physical Systems (CPS) is a knowledge source to update information on the blackboard.[10] Here blackboard showed real-time information on the status of other system/equipment. Each CPS use IoT architecture to communicate and update the required information to blackboard while this information can be seen by other CPS to act accordingly.

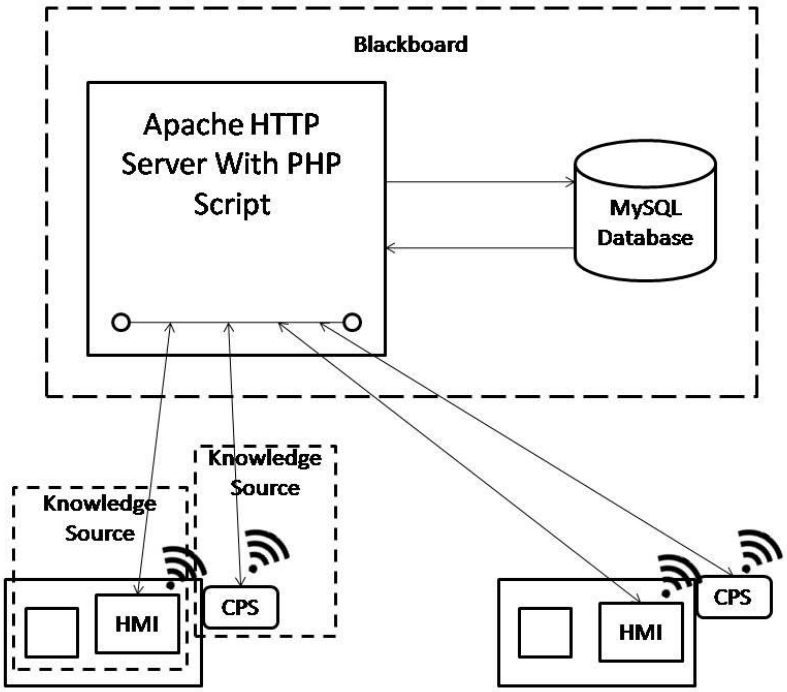


Figure 2.1 System integration using blackboard architecture

2.2.7 The Cloud

As seen in figure 4, the blackboard is used for information sharing between different entities. Blackboard act as common information sharing architecture. Such information sharing architecture can be implemented using the cloud. Also, many data analysis agents, digital twins and other third-party agents can be hosted on the cloud. An entity can send data to the cloud, algorithm present in the cloud can extract information from the data. Then this information is utilized for real-time decision making. Once

the decision is taken, then that is communicated to an embedded system present on failed to act accordingly.

2.2.8 Additive Manufacturing

Additive manufacturing is widely used for a small batch of customized products. Using technologies like fused deposition method (FDM), selective laser melting (SLM) and selective laser sintering (SLS), we can produce decentralised, high performance, cheaper and faster products that can be used for prototyping. Due to which time required for product development reduced drastically.[9]

2.2.9 Virtual Reality and Augmented Reality

Virtual Reality (VR) goals to generate a virtual world where the user interacts with these virtually generated graphics, while Augmented Reality (AR) goals to augment computer-generated graphics on real-world objects. Such as facilitating user with the current state of the machine. [11]Also, VR can be used to quickly find repair instruction with the 3D demonstration. [9]

2.3 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*".[12] 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 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.

2.4 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 the terminology used or in the understanding of the terminology used for the 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.

2.5 How standardization in industry 4.0 can help enhancing interoperability

As per [13], standardization plays a key role in the 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 facilitates training and specialized technical support.

3) Common security for the protection of business information, privacy of people, and all the information that is produced, transmitted, and stored.

A common language dictates the need for 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.

2.6 Need of multi-agent systems (MAS) approach

[14] explains the criteria set up 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: Motivation and Objectives

3.1 Motivation

The manufacturing industry consists of different machines made by various industries following various protocols. Also, some machines are communication capable, while some are legacy machines or manual machines that don't have communication capabilities. ERP software is a good means to keep customer orders, material, and human resources, but it fails to communicate with actual assets present on the shop floor. Lack of interoperability and decentralized decision making creates a void, causing ineffective utilization of resources and lack of agility to respond to changing customer need or any unplanned event.

ZVIE has listed criteria for the smart product, as discussed earlier. So, its use and demonstration in the context of Industry 4.0 are vital. To satisfy the ZVIE's smart product criteria, there has to be an architecture of communication, semantics for the communication, the way to virtual represent the asset, utilization of central services by the individual assets, format of standard function and security features.

As industry 4.0 revolves around decentralized decision making, each asset should take its own decision. To take the decision locally, each asset will need local computing power. As local decision power increases, the need for computation power also increases. There is need of individual function development which targets specific objective and provide intelligence to the asset agent. The work in past in agent architecture development focuses on agent architecture development for specific tasks that lack generalization. The same agent cannot be modified for other tasks easily with simple new function installation into it.[15][16][17]. The research in [18] focuses on generic multi-agent system development but in context with multi-objective optimization and lack functionality using which simple decision function can be added in the agent. The research work until now, as per the author’s knowledge, does not focus on developing a generic multi-agent system with functions and services that can be added to the agent as per requirement. So, there is a need for multi-agent architecture which absorbs and utilizes the function developed by different individuals. This leads to the development of a standard function structure that fits into multi-agent architecture seamlessly. Also, there will be reparation of computation at each asset which is a wastage of resources. Services are good solution where repeatedly required computations that are generic can be converted into a service. Each time asset required to perform specific computation, it calls a service that performs that specific computation. This can also help in an easy update of service at one place, which will affect all computations required by the assets.

All the discussion leads to the requirement of the development of multi-agent architecture along with standardization of function, services and digital twin to achieve specific objects while ensuring interoperability. Agents will enable the asset to communicate with the external world, which can be an administrative asset shell. At the same time, the digital twin of asset is a digital replica of an asset that is updated in real-time and respond to the real-world asset in real-time to achieve a specific objective.

In most cases, the digital twin will be a data model due to the availability of a large amount of data of environment and asset due to the large implementation of IoT technology. So, the use of technologies like ML, DL, RL or computer vision will ease the development of data models of assets. These digital twins with specific objectives can be transformed into function. The transformed functions can be installed into the agent, enhancing its capabilities.

3.2 Objectives

1) Develop multi-agent architecture to cater for the manufacturing ecosystem to achieve interoperability and decentralized decision making by introducing intelligence into the agent in the form of functions.
2) Utilization of standard language for the interactions with messaging protocols and using it to demonstrate intelligent interactions among machines.
3) Transform conventional 3-axis CNC milling machine into a smart machine using ZVIE's smart product criteria
4) Demonstration of communication between machine agents, between functions within the same agent and agent accessing services.
5) Develop a machine's digital twin for condition-based maintenance and then transform it in the form of function which a machine agent can utilize.
6) Develop function related to single machine sequencing with considering maintenance schedule and tool health along with early and late manufacturing penalties.
7) Demonstrate communication between machine agents, between functions within the same agent and between service and function
8) Use of Analytics viz. ML, DL, RL & computer vision to achieve overall operations planning.

Chapter 4: How to enable machines to interact

4.1 Basics of device interactions

For the basic transfer of messages between entities, these entities must be 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.

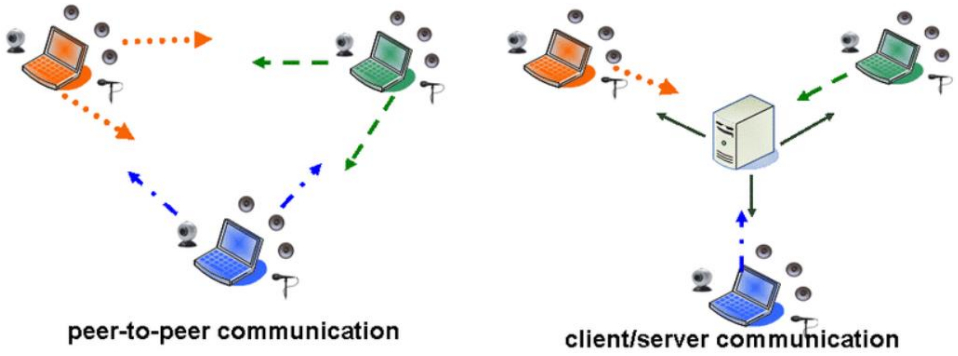


Figure 4.1: basic types of network model [google images]

The entities in a network must also adhere to some specific format(s) of data transfer that both sides should understand. And also, for complex communications, the contents of the messages should be such that both 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 message transport mechanism.

4.2 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 the 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 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 4.2: representation of JSON format

4.3 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 include 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 get transported using physical hardware protocols.

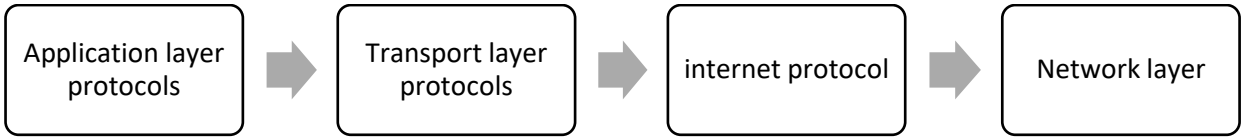


Fig 4.3: basic TCP/IP communication model

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

4.4 Messaging Language

The messages that get shared between the agents are written in a 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.

4.5 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 the 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 support for incorporating a standard language for these interactions.

Chapter 5: Messaging language

In essence, a language of communication in the 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. Two major messaging languages were developed and standardized for agent-based communications KQML and FIPA-ACL. Their descriptions are given below.

5.1 Knowledge Query and manipulation language (KQML)

KQML is an 'outer language' that defines an envelope format for messages that get transferred between agents[19]. KQML defines various 'communicative verbs' called performatives (E.g., Ask-if, perform, tell, reply, etc.) The performative along with the message content will constitute a speech act.

Fig 5.1: constituents of a speech act

example:

Performative**: Request**

content: "the door is closed"

speech act: "please close the door"

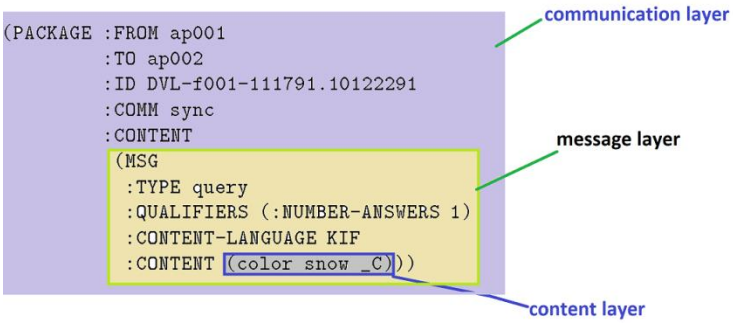


Fig 5.2: KQML layered structure

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: 5.3 Sample of a declaration message [9]

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.

5.2 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). The semantics of these performatives are given concerning 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 5.4: FIPA-ACL message

5.3 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 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.

5.4 Description of performatives

FIPA-ACL consists of 22 performatives as enlisted below

1. **Accept-proposal**: The action of accepting a previously submitted proposal to act.
2. **Agree**- The action of agreeing to perform a requested action made by another agent. The 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 the content of the request to ask another agent to tell us if 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 to 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 5.1: all FIPA-ACL performatives

5.5 Concept of the Interaction protocol

According to the FIPA-ACL standard, an interaction among machines is the exchange of performatives between machines or the Agents. But a combination of some performatives can result in different types of conversations. A predefined pattern of performative exchanges is called 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.

5.6 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 the manufacturing industry. Following is an explanation of those particular interaction protocols. The description and working of these protocols is explained in [20]

5.6.1 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 the request. If agreed, the agent that sends an agree performative is expected to inform with confirmation of completion of the action or with the result. In this case, the conversation ends with an ‘inform’ performative.

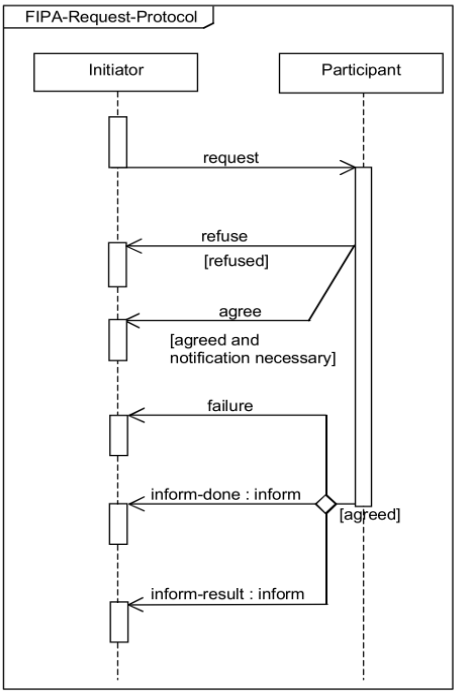


Fig 5.5: Interaction pattern of FIPA request protocols[21]

5.6.2 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 asking each agent to give a proposal following the 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 the best proposal. The agent that had sent the best proposal is sent 'accept proposals performative and the 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 the task to the initiator agent.

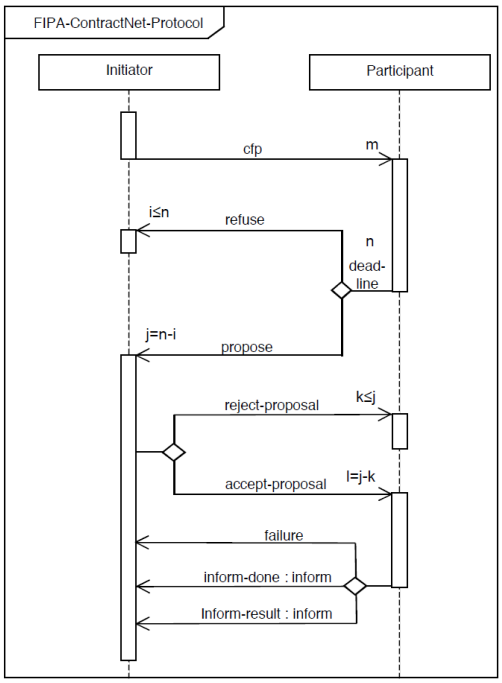


Fig 5.6: FIPA contract-net interaction protocol flow[22]

5.6.3 FIPA Subscribe Interaction Protocol

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

Explanation of the Protocol Flow

The subscribing process is simple. The 'subscribe' performative is sent to the agent that handles certain topics for a 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. Further updates on these topics are sent to the subscribers. To unsubscribe, "cancel meta protocol" will be used.

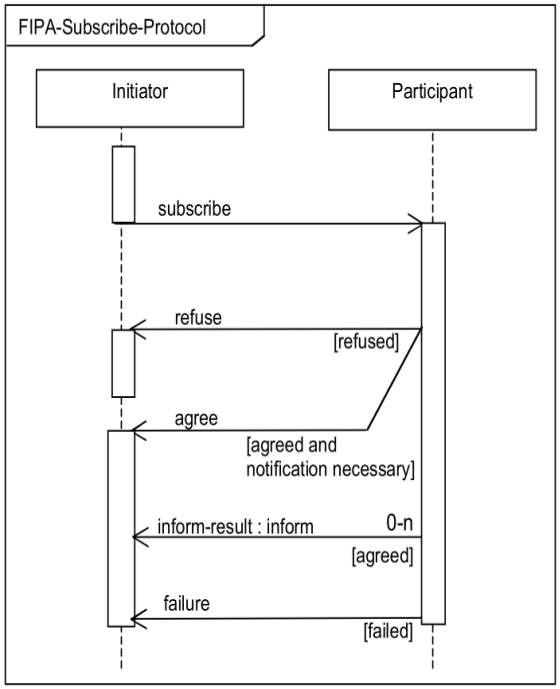


Fig 5.7: FIPA Subscribe protocol flow[23]

5.6.4 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

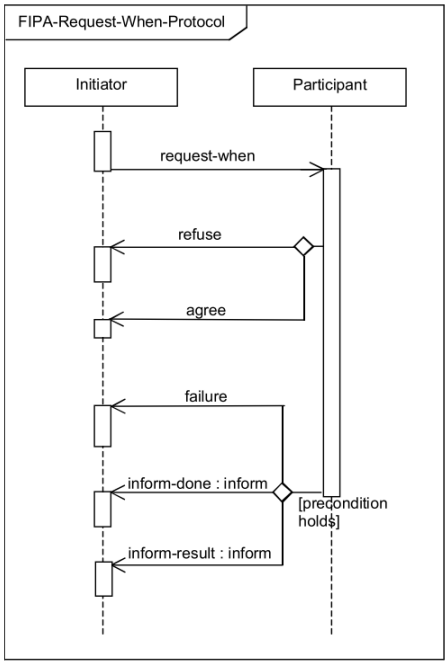


Fig 5.8: FIPA request-when interaction protocol flow[21]

Explanation of the Protocol Flow

The protocol flow is almost the same as the FIPA Request interaction protocol. The request when performative is sent with precondition in the specifications. The receiver agent will reply 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 the completion of the task.

5.6.5 FIPA Query Interaction Protocol

The query interaction protocol is used to confirm whether a certain proposition is true or not.

Explanation of the Protocol Flow

The initiation of the 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 about a particular object of interest. The rest of the protocol flow is similar to the request interaction protocol. Where the participant agrees or refuses to respond. If agreed, it is followed by an informed performative.

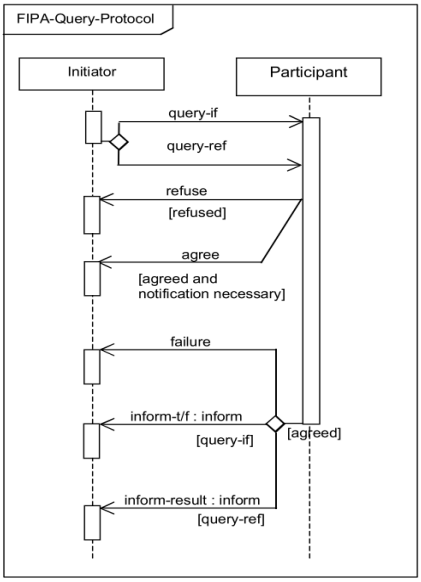


Fig 5.9: FIPA query protocol flow[24]

5.6.6 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 proposed communicative act when the receiving agent accepts the proposal.

Explanation of the Interaction Protocol Flow

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

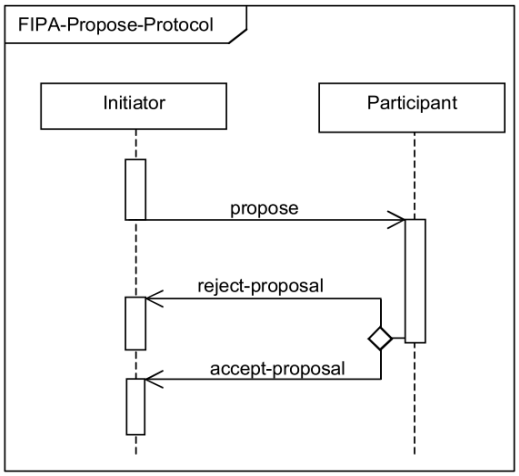


Fig 5.10: FIPA Propose Protocol flow[25]

5.6.7 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 the 'reply by parameter.

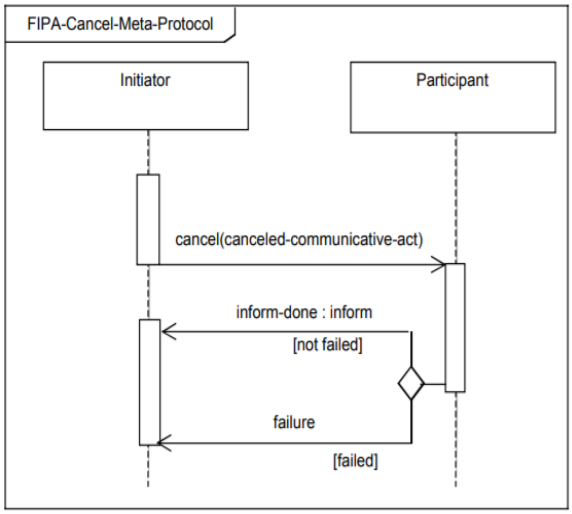


Fig 5.11: FIPA Cancel-Meta Protocol flow

5.6.8 FIPA recruiting interaction protocol and FIPA broking interaction protocol.

The recruiting protocol and broking protocol are 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 knows to figure out who is its target agents, that case the proxy agent is sent a list of target agents. However, when an 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 5.12: 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 a message will include a 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 brokers 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 a certain designated receiver as specified by the initiator.

5.7 Concept of performative types

Performative types do not have a mention in FIPA works of literature. These have been introduced in this project. A performative type is a piece of 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 the request are to be understood from the type and content of the message. For this purpose, there is a need for explicitly specified types that each machine understands. The types are linked to a specific performative only. The 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 the 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 concerning 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 are different. A machine may tell the material handling entity to inform when there is a failure in transit, so in this case, the machine uses request-when performative and failure in transit as a type.

But in some other cases 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 to a particular topic)

Table 5.2: examples of performative types

5.8 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 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	3rd 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 the conversation.	
Ontology	Content ontology	
Reply_by	Reply within a particular amount of time	

Table 5.3: 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',
}'
```

Fig 5.13: outer envelope syntax for the FIPA-ACL message (JSON format)

5.8.1 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 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) that 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 the previously sent message to which this message is a reply. This parameter will be blank when the 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.

5.9 Case example for how FIPA interactions take place

The following example explains how FIPA-ACL contract-net protocols are used for interactions within machines. This is a basic example of 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 the machine (M1) has finished processing 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, a conversation using contract-net interaction protocol is initiated and the machineM1 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.

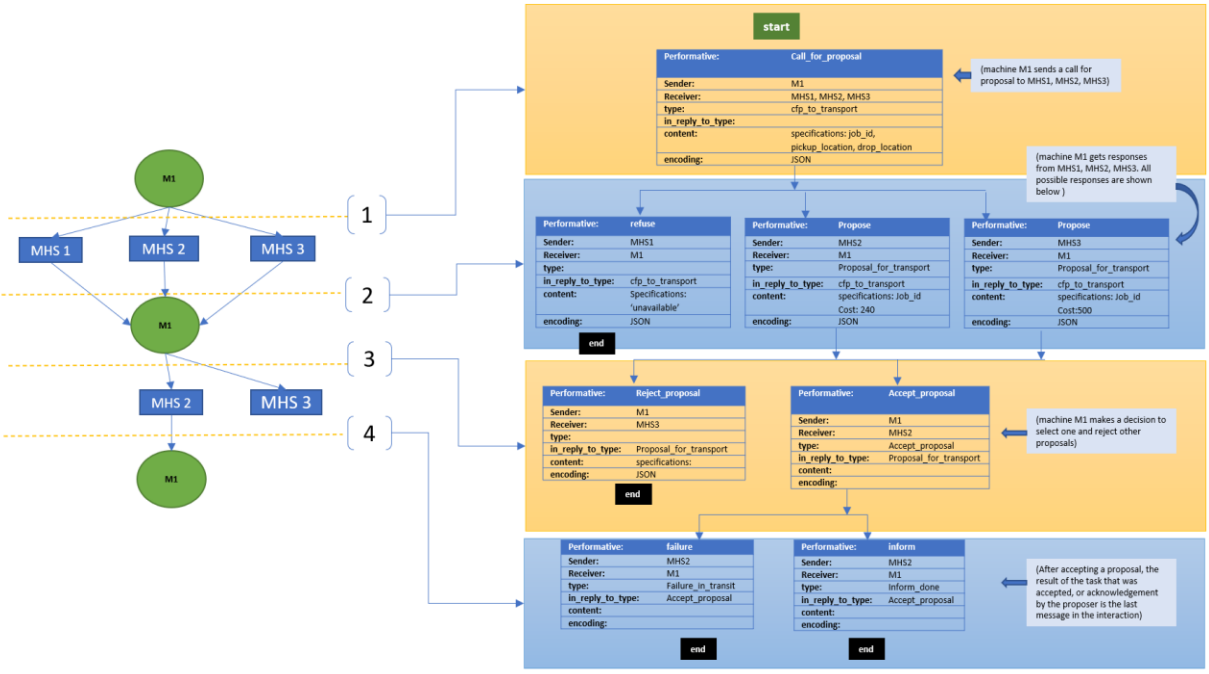


Fig 5.14: example for contract-net interaction protocol

At the second stage of interaction, based on their current location and specifications of pick up 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 refuse performative with some reason and the other two MHS propose a cost.

In the third stage of the interaction Machine, M1 decides whose proposal was the best and therefore which particular proposal to accept based on cost, the 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 the task, the MHS2 will respond with inform performative with 'inform_done' type.

Chapter 6: 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 facilitates messaging amongst Agents.

6.1 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.

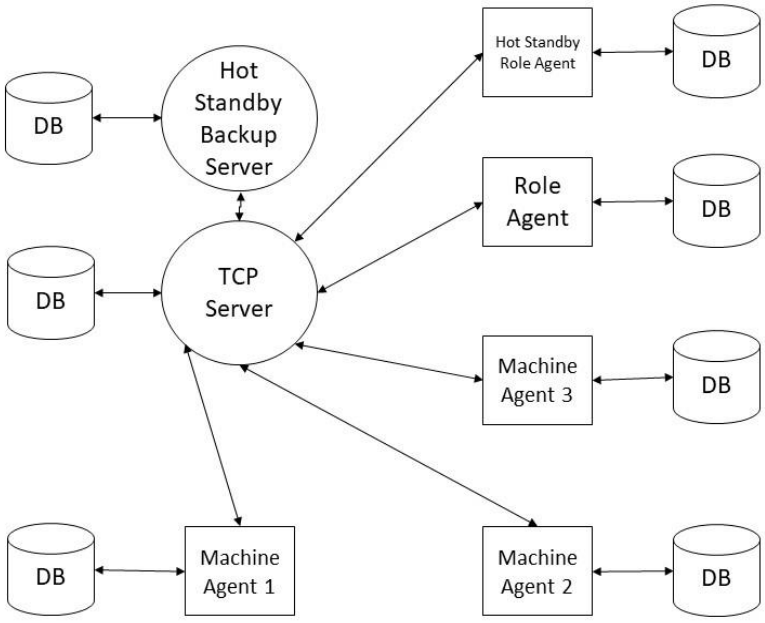


Fig 6.1: Proposed Multiagent Architecture

6.2 Web-sockets based client to server communication

A network protocol ensures the connectivity and transfer of messages between servers and Agents. WebSocket protocol is a standard computer communication protocol that 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.

6.3 Agent description File

The agent description file or agent file consists of agent name, Agent role, and abilities to understand 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 mean that the outputs of the functions that handle these performative types should have those particular specifications mentioned in the 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 6.2: An Agent file

6.4 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 secure 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, the learning process can be shifted to the cloud or local server in the form of a service. The updated trained model can be downloaded from time to time to the agent to take local decisions 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 7: Server of multi-agent system

7.1 Server network access

The server and agents should be connected to 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 the HOST IP ADDRESS and PORT number on which this server will reside.

Currently, in the 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.

7.2 server initiation, handling new initiated agents

The code for the 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, 2 major functions 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 the 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 set up 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 AGENT NAME 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.

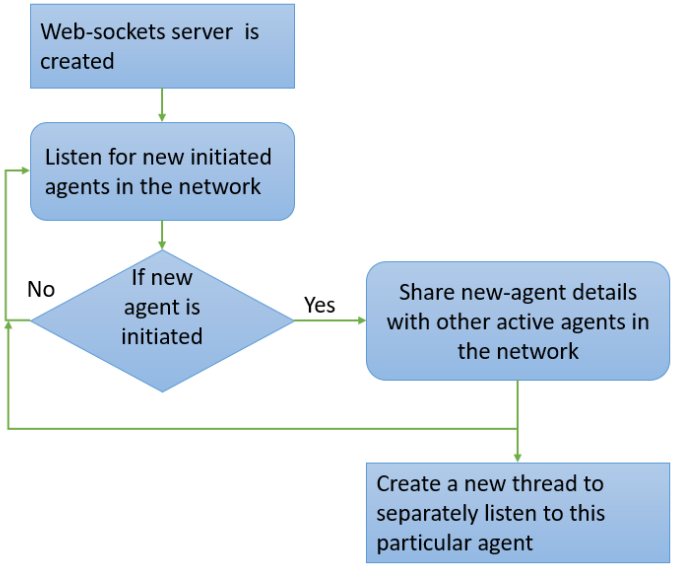


Fig 7.1: server initiation and handling new initiated agents

7.3 How server handles messages from the Agents

The 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 the server itself, then those messages are handled separately by the server.

Chapter 8: 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.

8.1 Machine Agent architecture

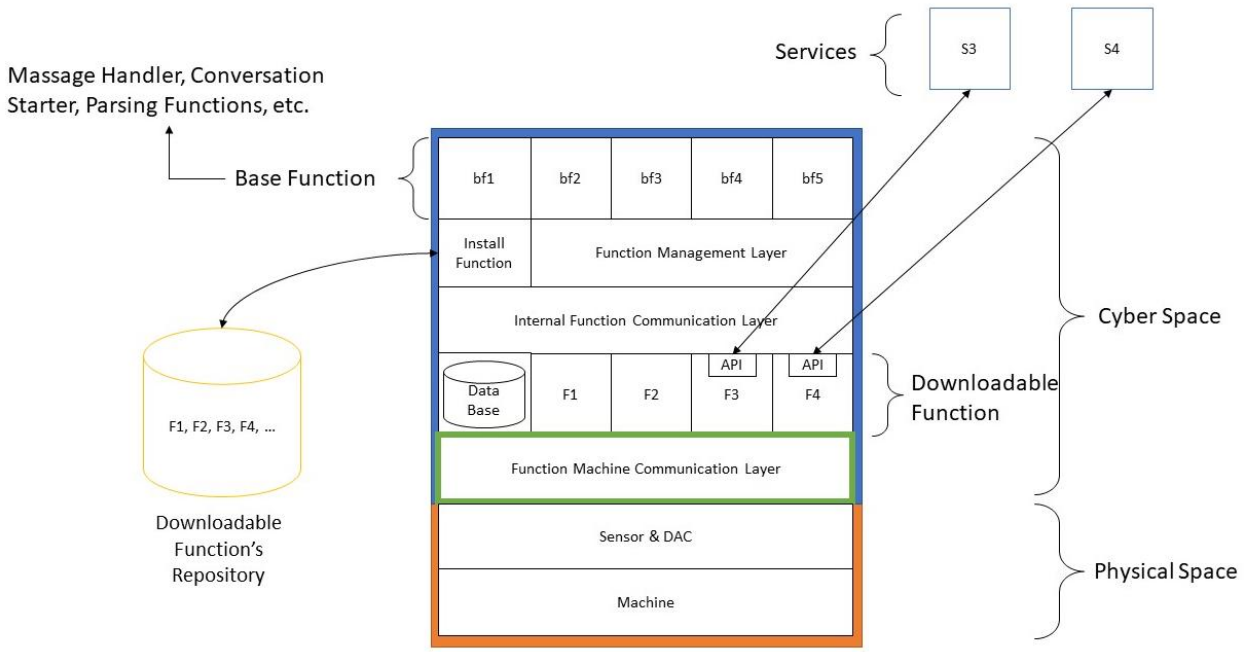


Fig 8.1: architecture for a machine agent

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 sensors 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 commands 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 functions. 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.

8.2 Role Agent and its architecture

The overall architecture of a role agent is similar to that of a 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 agents is similar to that of a machine agent. After installation, the user has to download those specific role agent functions into the agent, which is meant for a specific shop floor functionality.

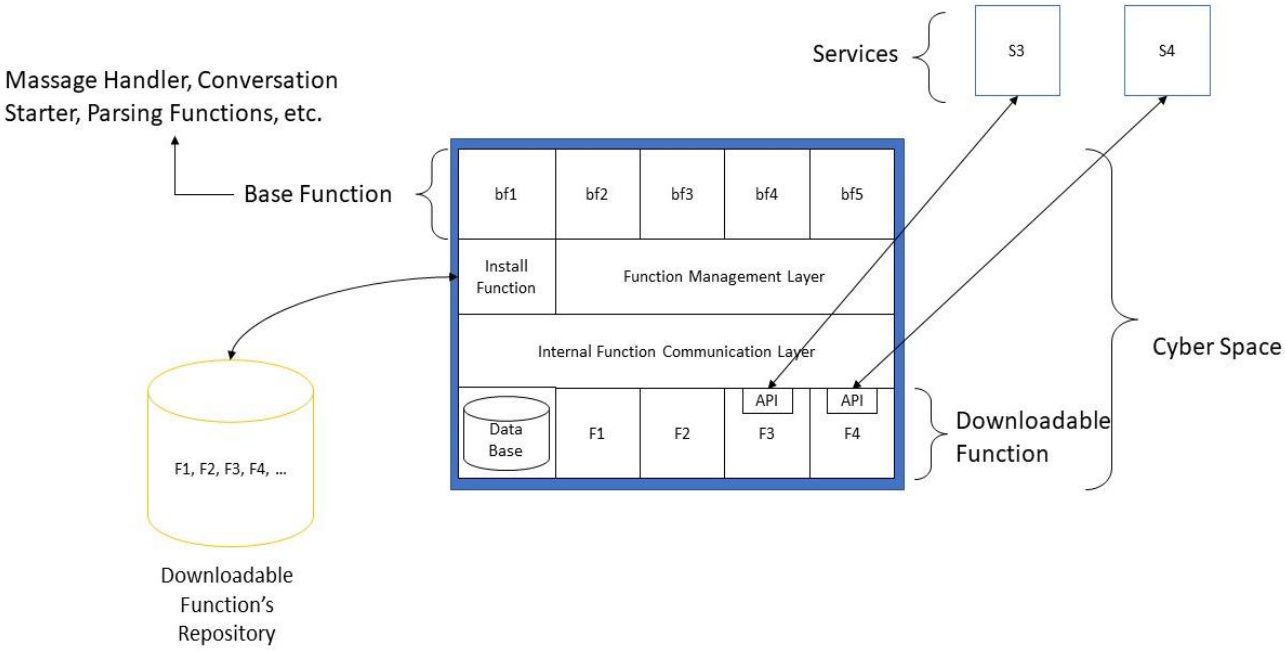


Fig 8.2: architecture for a role agent

Consider user has installed an agent, and now he/she wants 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 jobs 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 to pass information to all the agents. When the customer places an order, using which batches of jobs are planned. These planned batched 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 agents 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 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 computational hardware to reside on.

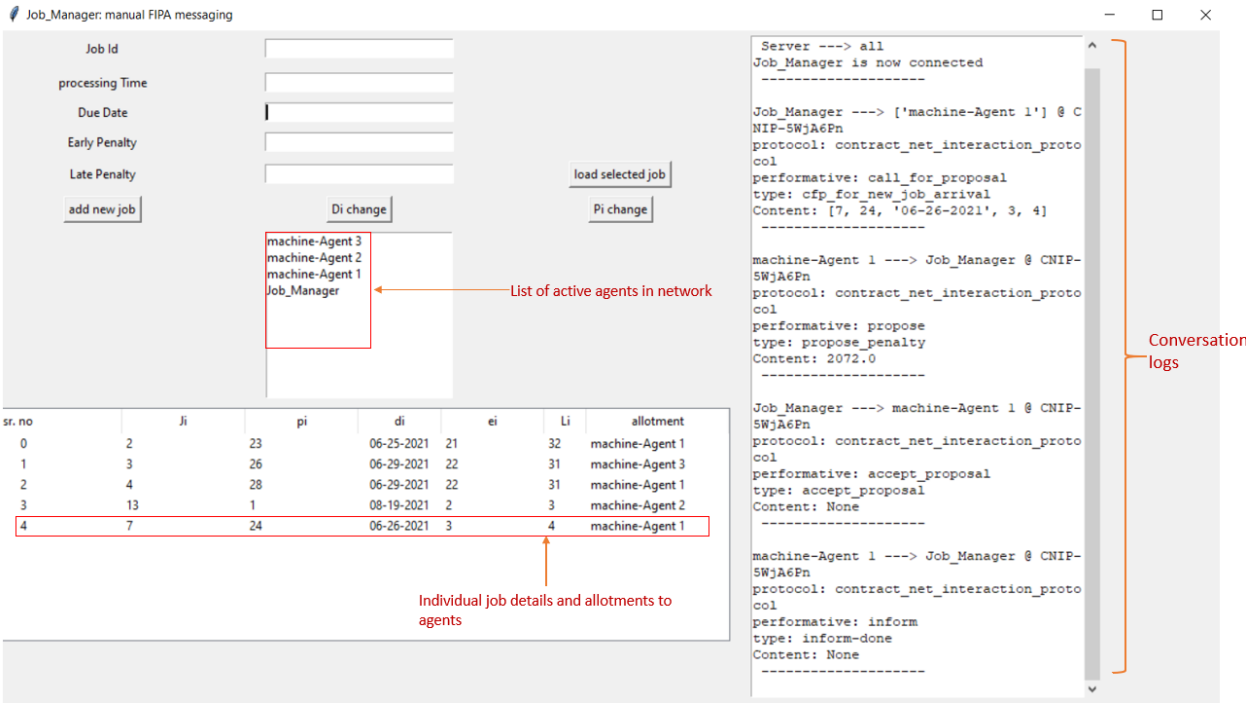


Fig 8.3 GUI for job manager agent

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 a new job has to be added there. As soon as the operator clicks on the “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 assesses 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 the allotment, the manager displays the new allotment of the job on the screen along with its details.

The button “load selected job” is used to fill in 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.

8.3 Agent functions

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

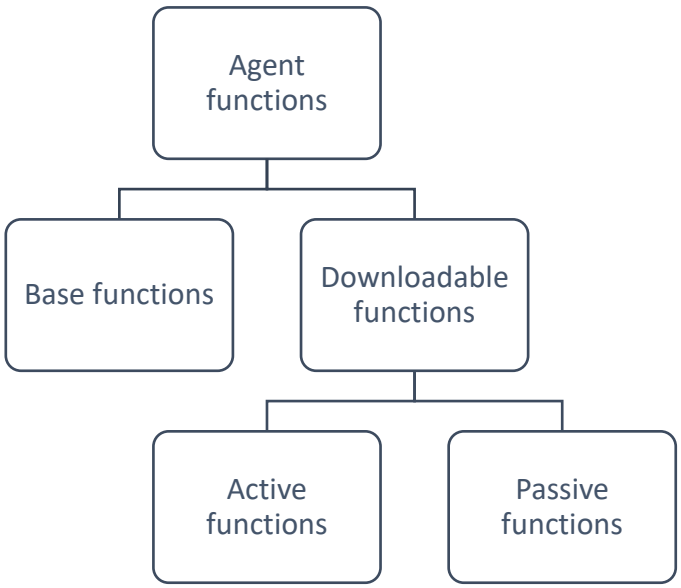


Fig 8.4: abbreviations of functions and their hierarchy

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. The name of each function must 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.

8.3.1 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.

8.3.2 Downloadable Functions

These functions do not come with the 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.

8.3.2 a) Active Functions

Active functions are those function which runs continuously doing some dedicated task. Whenever an agent is initiated, at the time of starting each active function acquires a separate thread on the processer 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 an

active function could be a condition-based maintenance function that continuously monitors the sensor data and predict the 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 functions exists called passive functions.

8.3.2 b) Passive Functions

These are a class of functions that respond only when called. These functions accept input in a particular format and produce 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.

more optimal sequencing decisions. Some downloadable functions may have the corresponding service associated with them.

8.4 Function Type

There could be a number of types of function. Each downloadable function has a type 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 standardised. 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.

8.5 Hierarchy of Functions in the Agent

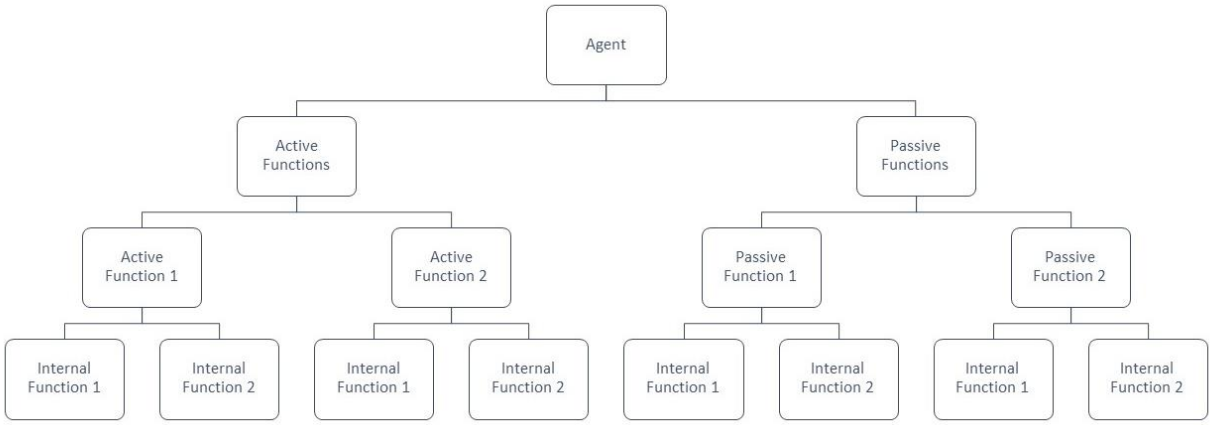


Figure 8.5 Hierarchy of active and passive functions in the agent

The agent is made up of a function with a hierarchal structure, as depicted in figure 13. When we divide functions into active and passive functions, we see a hierarchal structure, as shown in figure 13 below. There could be multiple active and passive functions in the agent. These functions are made up of internal function. We call a ".py" file as a function. That could belong to an active or passive category. Inside this ".py" file, there could be a lot of functions written with python syntax "def function_name():", we call each of such function inside the ".py" as an internal function.

When we divide functions into the base and downloadable functions, we see a hierarchal structure, as shown in figure 14 below. There could be multiple base and downloadable functions in the agent. These functions are made up of internal function. We call a ".py" file as a function. That could belong to a base or downloadable category. Inside this ".py" file, there are number of functions written with python syntax "def function_name():", we call each of such function inside the ".py" as an internal function.

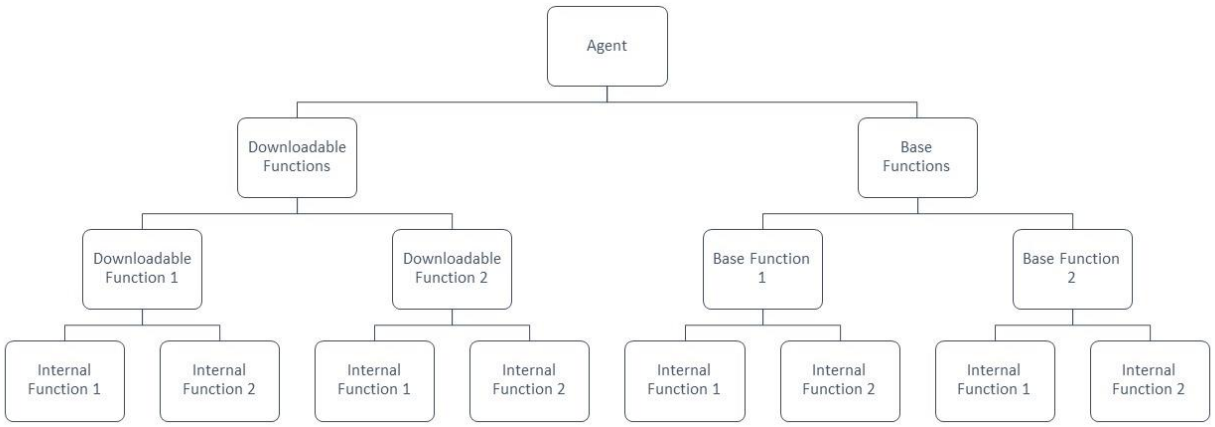


Fig 8.6 Hierarchy of downloadable and base functions in the Agent

8.6 Structure of Passive Downloadable Function

Figure 15 shows a proposed structure of the passive downloadable function. It has a number of internal functions. At the start of any function file, we have to import the required libraries and other downloadable function. Which downloadable functions to import can be known by accessing the FunctionDependency.txt, which is one of the configuration files.

One of the internal functions is the introduction internal function. This internal function aims to provide basic information about the function like its type, whether it is an active or passive function, its dependencies, its communication capabilities, etc., when called upon. It is mainly used during function installation.

In the case of a passive downloadable function, execute internal function is used when there is inter-agent communication is underway, and it is not used when there is a downloadable function to function communication within the same agent. In an example of agent to agent communication, agent 1 wants to know the health status of agent 2. Then agent 1 will initiate the conversation with a specific performative. The message will reach agent 2's message handler function. Now, as per the configuration message handle function will pass on the information to one of its downloaded function related to machine health status. But now message handler function does not know which internal function to call, how they are internally structured, etc. So, it calls to execute the internal function and passes the performative and the information got from agent 1. Now it's the duty of execute internal function to call required internal functions according to performative and information received, generate the required output and pass it

back to the message handler. Here “execute” internal function acts as an I/O for the downloadable function.

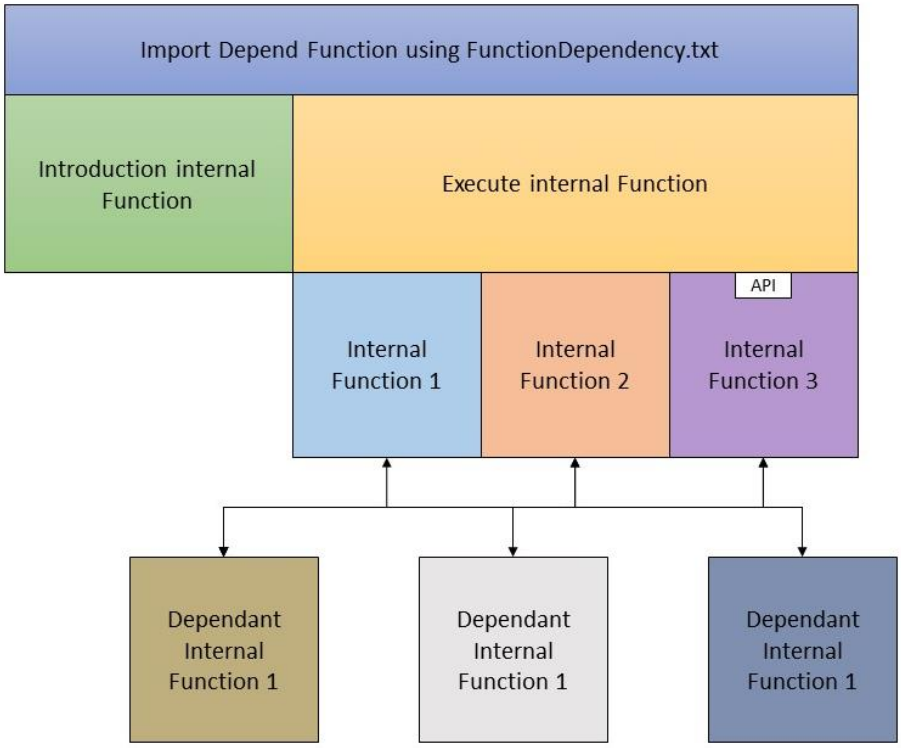


Fig 8.7 Structure of passive downloadable function

As depicted in figure 15 Internal Function 1, Internal Function 2, Internal Function 3, etc., are internal functions of passive downloadable function. An example could be a single machine sequencing function file having an internal function, one for calculating early and late due date penalty, one for a job assignment in the queue, one for resequencing of the queue, etc. Which function to call is decided by execute internal function based on performative and information it receives.

There could be a case where the internal function of the passive downloadable internal function calls an internal function of another downloadable function. In such cases, they call each other directly without calling the execute method or “execute” internal functions, as depicted in figure 15. Also, some internal function could utilize the services present on the cloud or server within the network through APIs.

8.7 Structure of Downloadable Active Function

Below figure shows a proposed structure of the active downloadable function. It has mainly three internal functions, i.e. introduction internal function, active internal function and callable internal function.

One of the internal functions is the introduction internal function. This internal function aims to provide basic information about the function like its type, whether it is an active or passive function, its dependencies, its communication capabilities, etc., when called upon. It is mainly used during function installation.

In the case of an active downloadable function, it should run continuously, doing its intended job. In the initialization of the agent, the main program file will run the active internal function of the active downloadable function on a separate thread. At the start of any active internal function, we have to import the required libraries and other downloadable functions as given in the FunctionDependency.txt, which is one of the configuration files.

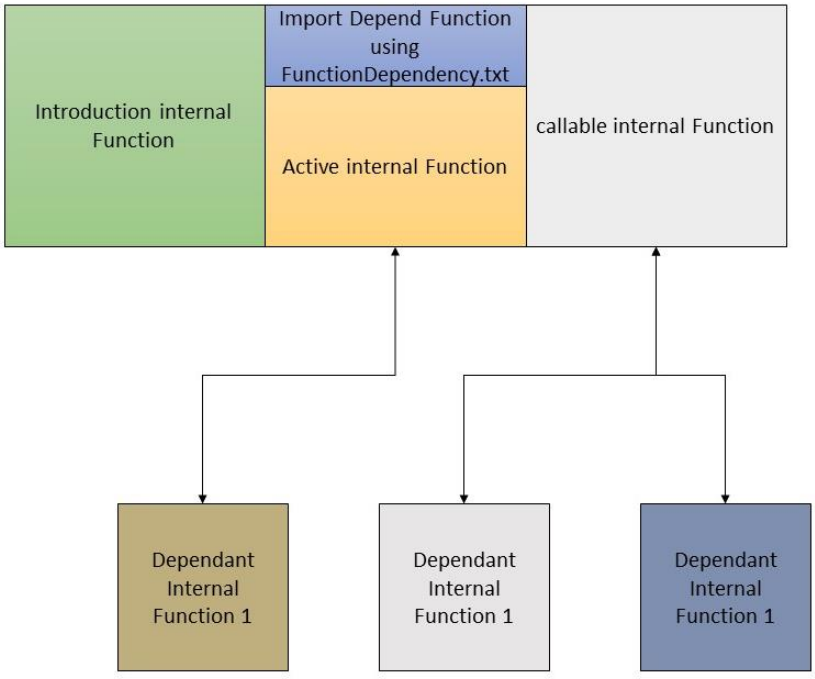


Figure 8.8 Structure of downloadable active function

When called on a separate thread, this active internal function runs continuously, doing its intended job—for example, the tool health monitoring. The “tool health monitoring” function has an active internal function that continuously reads live sensor data via the machine communication layer and produces output that it saves in its database. When other function within the agent wants to know the current health of the tool they call the callable internal function which inherits the properties of the passive function. The callable internal function reads the current status of the tool from its database maintained by the active internal function and returns the value.

As depicted in figure 16, the active internal function can utilize the other function to completed its intended work, while the callable internal function is called by other functions when they want to utilize this active downloadable function.

8.8 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 resequencing its job queue as a new job added to its queue. Resequencing is a computationally heavy job, so we can create a service that sequences 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 resequencing 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.

8.9 Agent File 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 figure 17.

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 "BaseLibraries" 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 "BaseLibraries" directory are ".py" files.

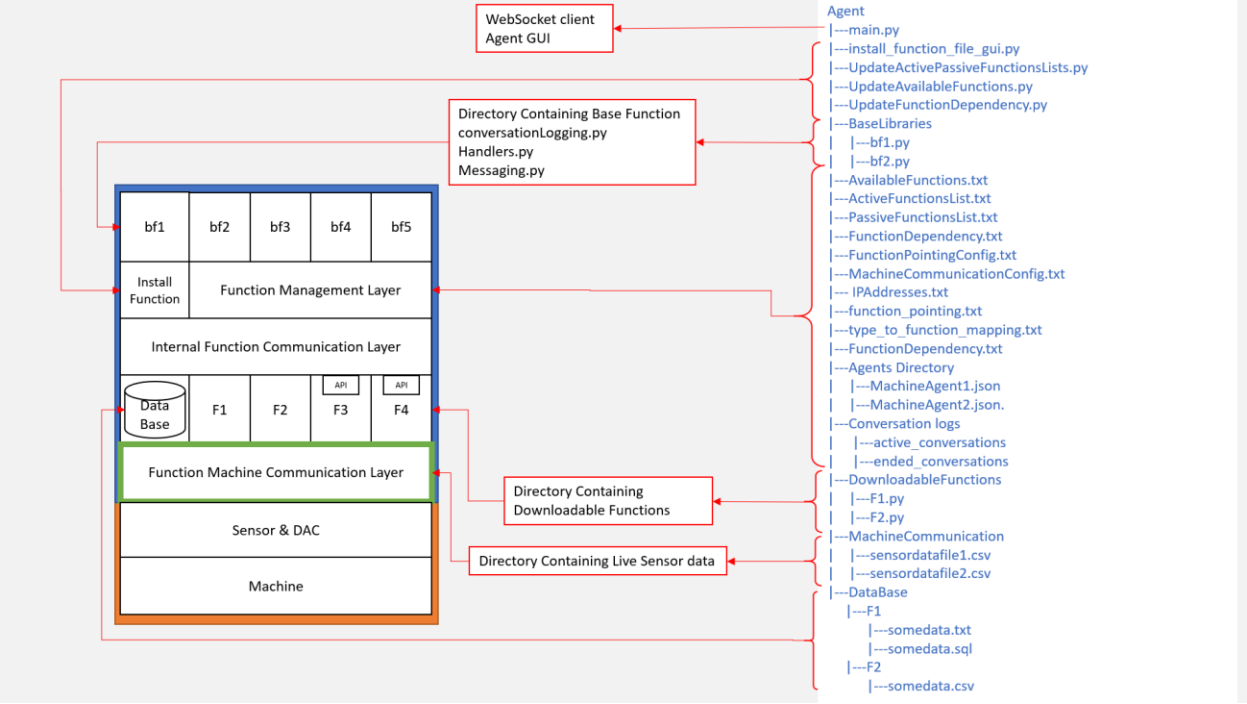


Figure 8.91 Agent file system

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 creates its own

directory in the Database directory and 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 the "Database" directory in ".txt" format while function2 save its metadata in the same directory in the ".csv" format.

The "MachineCommunication" directory represents the Function Machine Communication Layer of the agent architecture. DAC system present in the machine dumps live sensor data in this folder. The downloadable functions can utilise the data present in this directory for the analysis. This directory act as an interface between downloadable functions and machine DAC system.

The "main.py" file is the main program file of the agent. To initialise, the agent user needs to run this file. It holds the main control of the agent. The agent initialisation 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 utilised 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.

8.10 Function Installation

To install any function first, it needs to be downloaded from some repository like 'Github' or any other platform. A separate program file named "install_function_file_gui.py" is available in the agent to install the functions in the agent. When the user runs this file, it opens a GUI where the user can select the downloaded function from the repository. The installation program will automatically copy it to the “DownloadableFunctions” directory. Then it calls "UpdateAvailableFunctions.py", "UpdateFunctionDependency.py", and "UpdateActivePassiveFunctionsLists.py" files sequentially.

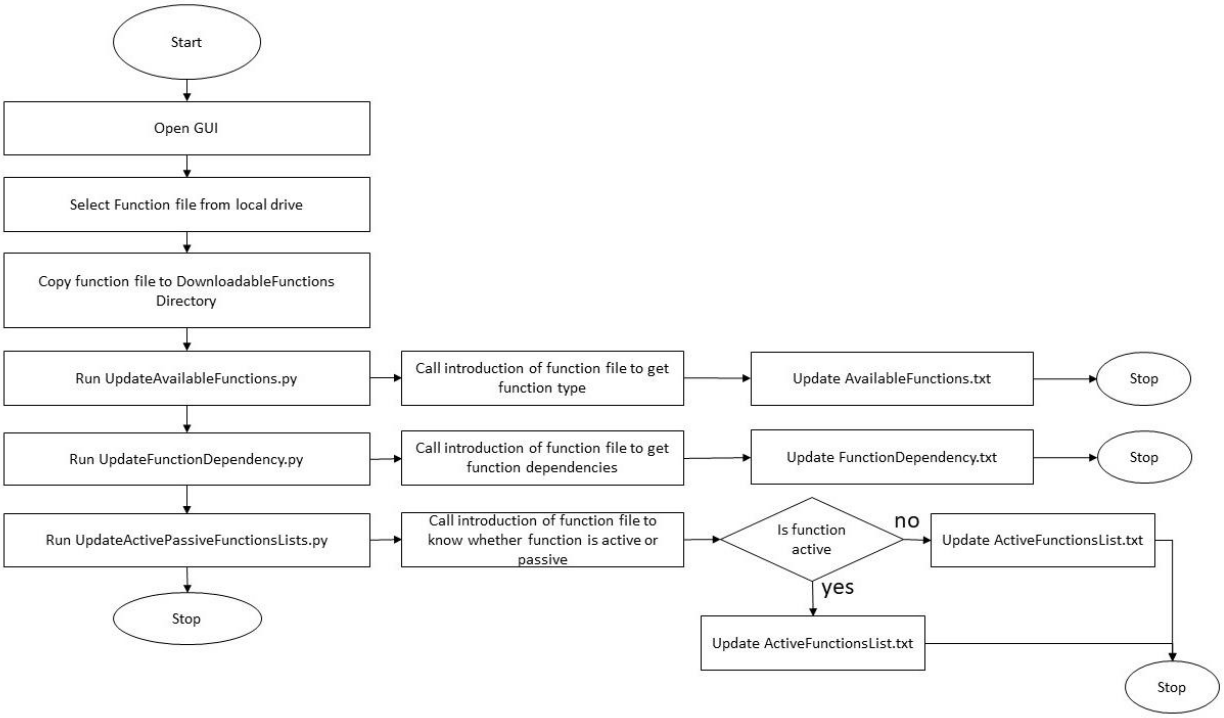


Figure 8.10 Function installation flowchart

The "UpdateAvailableFunctions.py" file, which calls the introduction internal function of the copied function file to know the function type, and it updates the “AvailableFunctions.txt” file. The “UpdateFunctionDependency.py” file, which calls the introduction internal function of the copied function file to know the dependencies of the function, and it updates the “FunctionDependency.txt” file. The “UpdateActivePassiveFunctionsLists.py” file, which calls the introduction internal function of the copied function file to know whether the function is active or passive type. If the function is of the active type, it updates the “ActiveFunctionsList.txt” file, or if the function is of the passive type, it updates the “PassiveFunctionsList.txt” file.

8.11 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 functions 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.

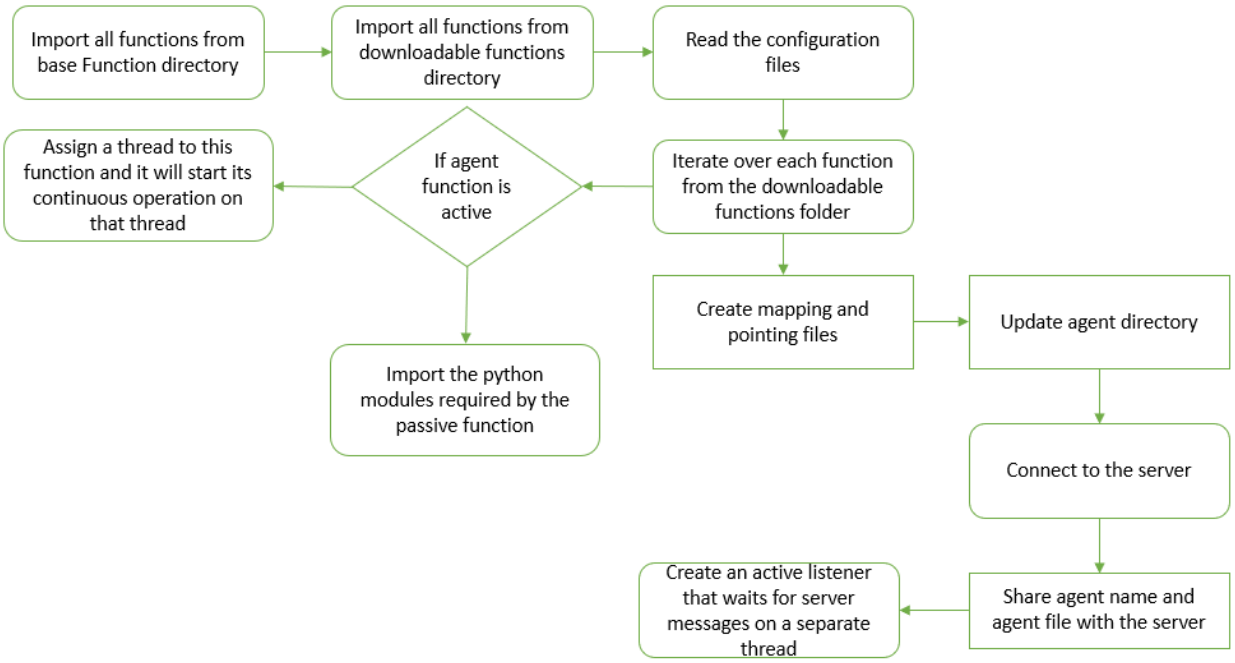


Fig 8.11: Tasks at agent initiation at the individual system level

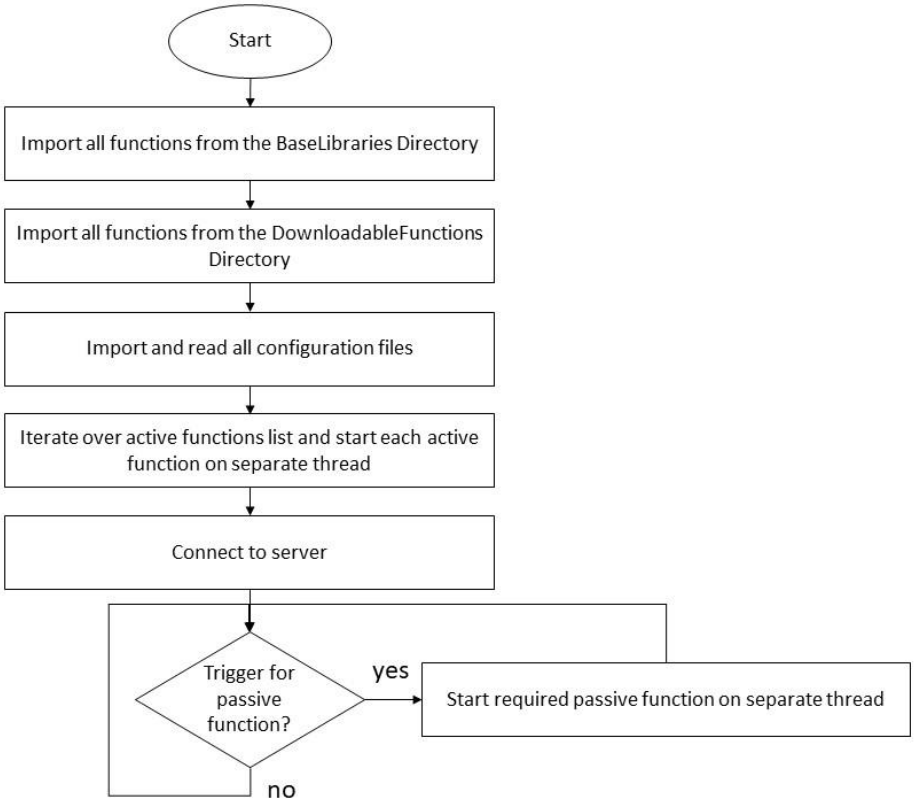


Figure 8.12: Agent initialisation flowchart

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 "BaseLibraries" and DownlodableFunction 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.

8.12 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.

8.13 Function classification

Each downloadable function serves a certain function. This function either tries to detect triggers or respond to messages posted by other agents. But there can be multiple numbers of functions that can serve the same purpose. For example, if an agent wants to calculate the RUL for the 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 functions would be CBM. All functions of CBM type should accept input and produce output in a specified format which needs 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 the 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.

8.14 Inter Function Communication

When an agent is initialized, the main program imports all the functions, which means all functions are also initialized. In the initialization process, the functions with dependencies, i.e. which depend on other functions for certain computation, read the “FunctionPointingConfig.txt” file and import the pointed functions.

Let's consider an example, “Function1” is of “abc” type and requires the help of function of type “xyz”. There are two functions, “Function2” and “Function3”, with type “xyz” installed in the agent. In the “FunctionPointingConfig.txt”, it is pointed that “Function1” should use “Function3” when it requires help from “xyz” typed function. So when “Function1” is initialized, it imports “Function3” and not “Function2”.

As “Function1” of type “abc” is dependent on functions with type “xyz” it knows the internal structure and how to call any internal function of type “xyz” with standard input. It also knows what output it can expect from function type “xyz”. This all information is hardcoded in the function type “abc” which is dependent on function type “xyz”. So “Function1” is capable for communicating with any function with type “xyz”, in our case “Function3”. If “FunctionPointingConfig.txt” file is edited and if “Function1” is pointed towards “Function2”, “Function1” is equally capable of communicating with “Function2” due to its type.

Here comes a need for the standardization of input and output for the function types. All the functions of the same type should accept the input and produce output in the same standardized way.

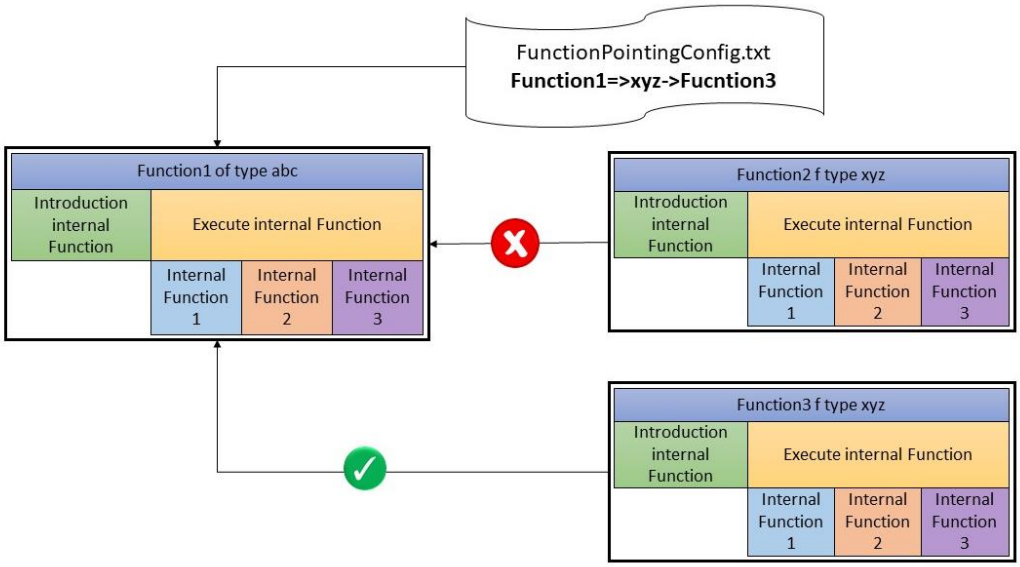


Figure 8.13: Inter function communication schematics

8.15 Major base libraries or base functions of an Agent

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

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

1. flatten message: this would take a JSON file and serialize it to a 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 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.

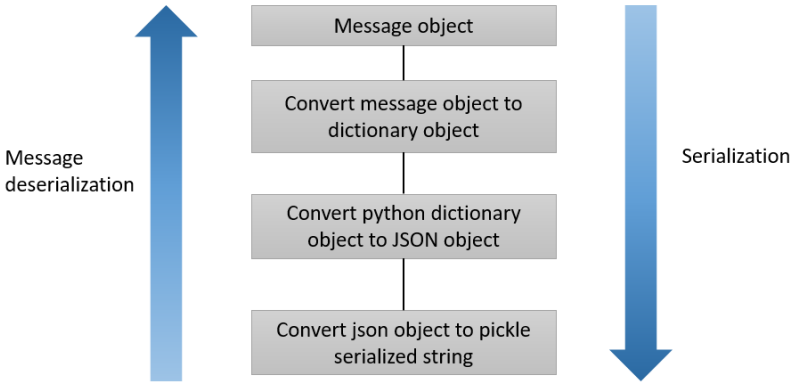


Fig 8.14: Serialization and deserialization of a message

8.16 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 the parent message handler, first, it is checked whether the message is a 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.

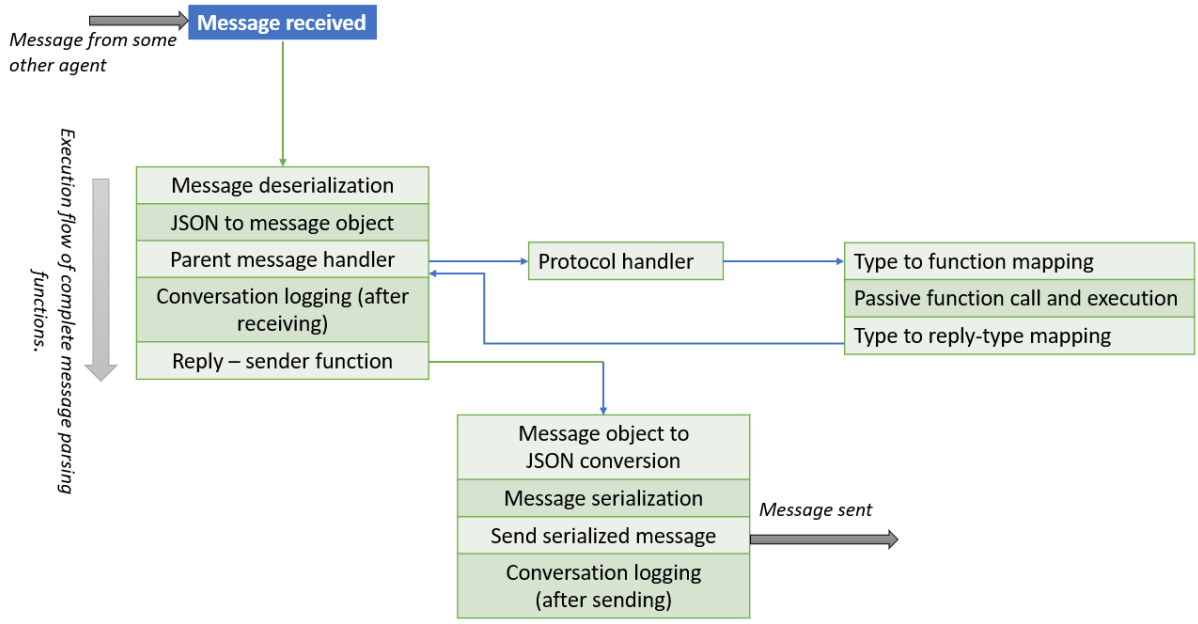


Fig 8.15: All message parsing functions

8.16.1 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.

8.16.2 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.

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 “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 that 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.

8.17 Trigger identifications

The trigger identifications are done via the active functions. But to keep the 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.

8.18 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.

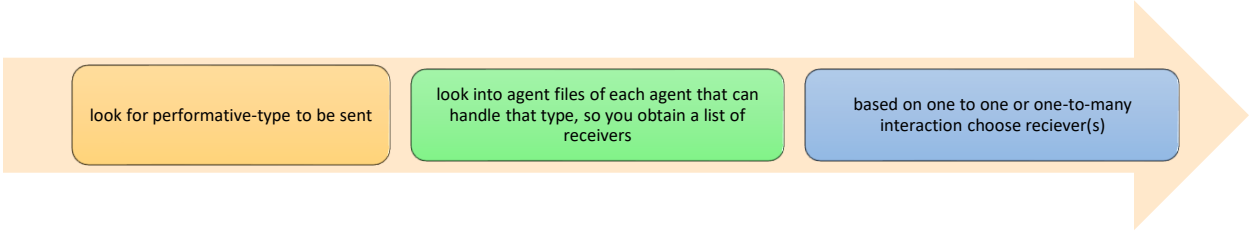


Fig 8.16: deciding target agents by target agent identifier

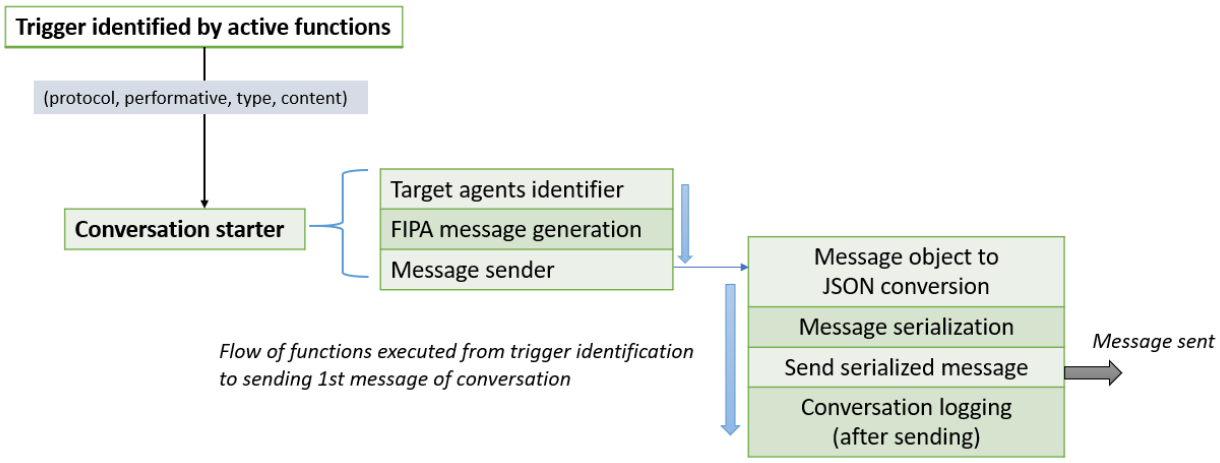


Fig 8.17: Conversation starter function and other base functions that it uses

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

8.19 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. In 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.

8.20 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 has subfolders 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 is different. There are three major functions for logging and their tasks are as follows.

1. The first function is to create a JSON log file. Mostly used by an agent for logging the 1st message of conversation that is being sent.
2. The second function is to log and update a preexisting log of the 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. The third function is to update the pr-existing log and end the conversation. This means that the 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 is named after the conversation-id. For example, “RIP1” is the conversation -id for request interaction conversation, and its contents are mentioned below.

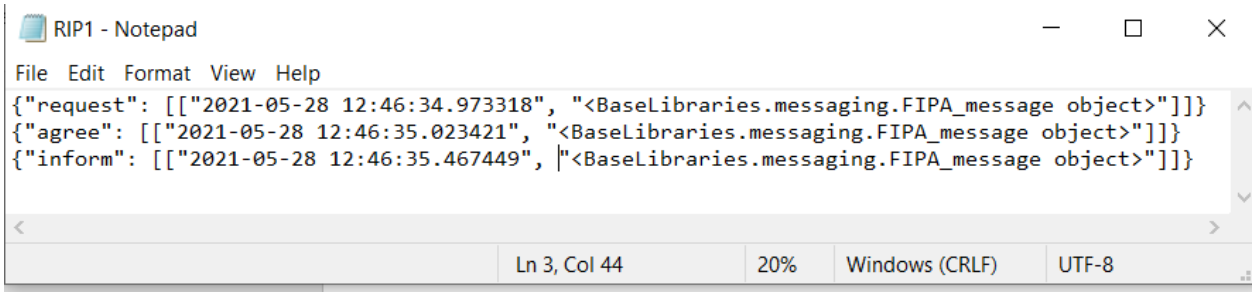


Fig 8.18: conversation log for a completed conversation that took place with request interaction protocol

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 decide to end the conversation after some time.

Chapter 9: Integration of agent functions into the agent

9.1 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 their functionalities. The active and passive functions govern how the agent behaves. The installation process of such functions is easy and is mentioned in the function-installation section. These agent functions are stored in a folder named “downloadable functions”.

An active or a passive function is 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”.

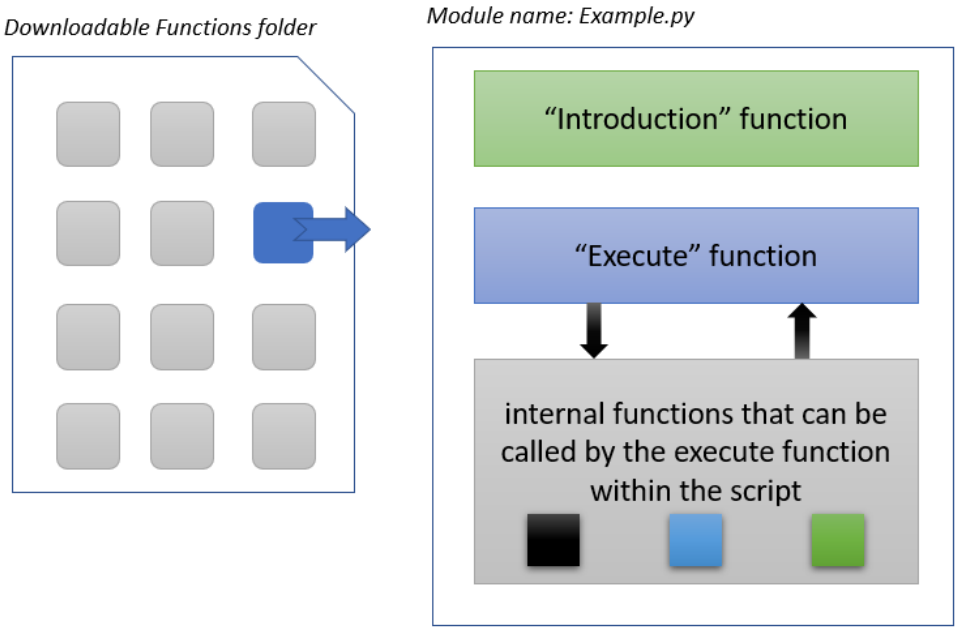


Fig 9.1: basic Structure of a downloadable function

9.2 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 the “DownloadableFunctions” directory, which contains all the downloadable functions 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 come 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, the folder can contain mixed format data files created by different functions. For example, function1 saves data in the “Database” directory in “.txt” format while function2 saves 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.

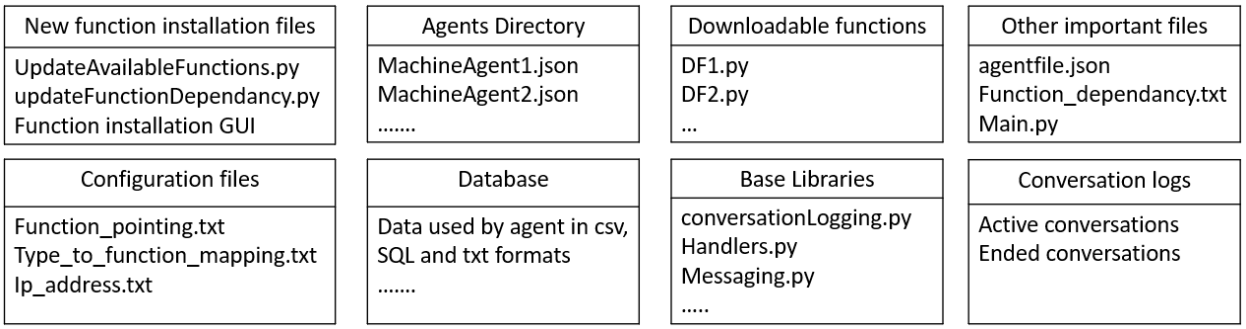


Fig9.2: Agent folder file system

9.3 Generation of mapping and pointing files

mapping (performative-type to function-type mapping) means to select a category or a class of functions called “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.

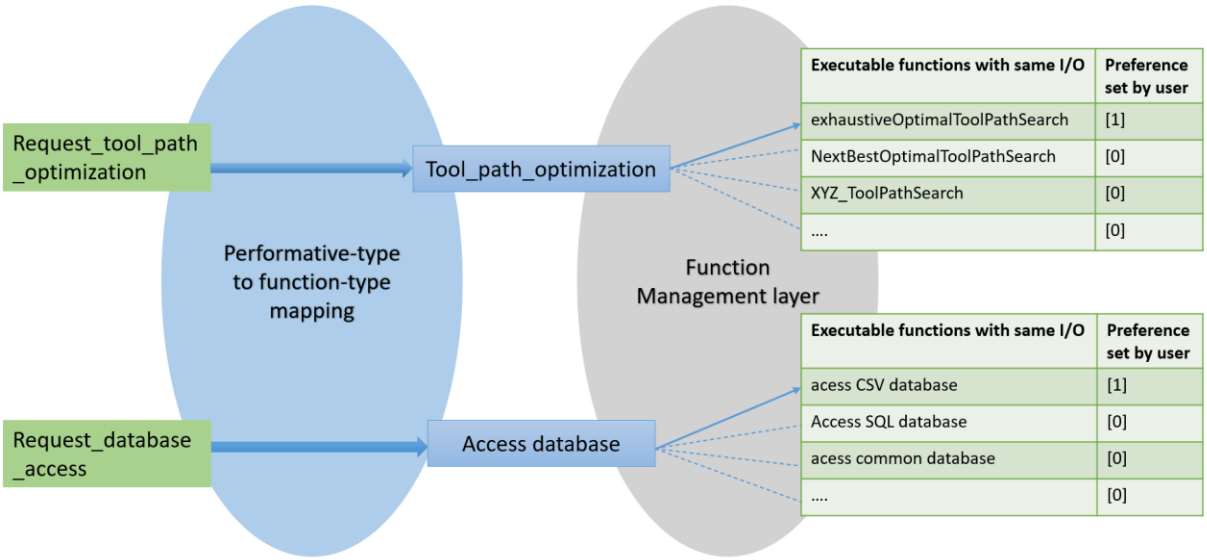


Fig 9.3: Mapping and pointing operation

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.

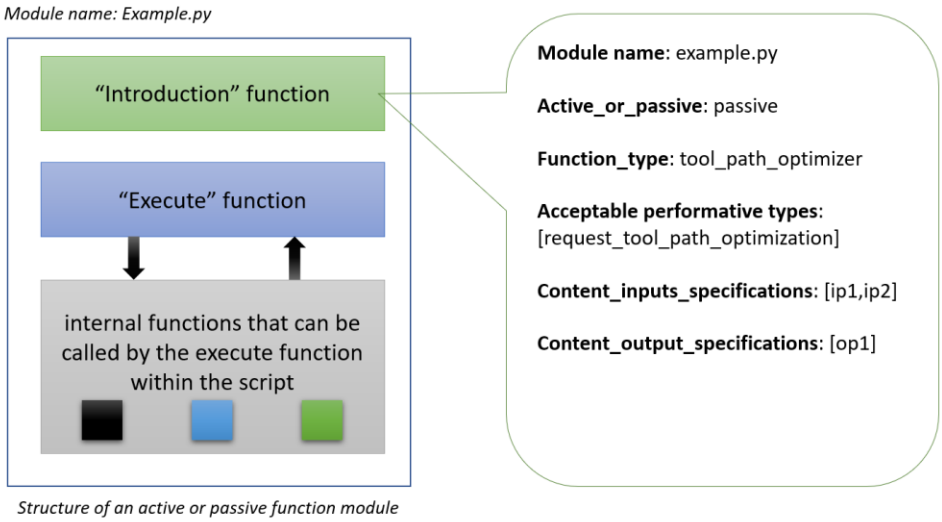


Fig 9.4: description of introduction function (internal) of a downloadable function

Reading the introduction part (internal function) of each function when the 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 are mapped using the "=>" sign. As shown in the figure below.

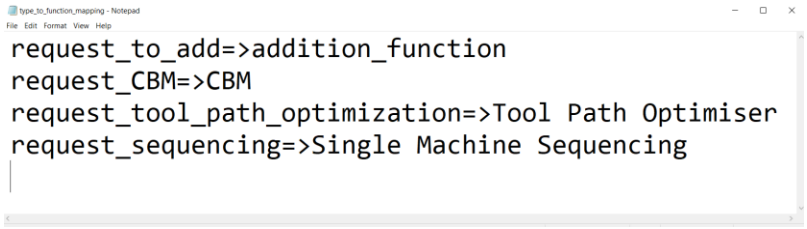


Fig 9.5: function-type to function-mapping file

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

The agent 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 output as required for a particular performative type.

9.4 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 are called the function management layer of the agent architecture.

The 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.

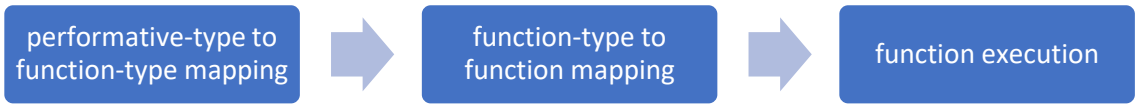


Fig 9.6: the process of selecting an executable function

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.

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

The "AvailableFunctions.txt" file contains the list of all the downloadable function installed in the agent. It is structured as shown in figure 18 below. Let's say the user installed a function named "add_sub" of type "BasicMathOperation"; then it will appear in the file in the format:

BasicMathOperation=>add sub

If the user installs another function named "mul_div" of the same type "BasicMathOperation"; then the file will be updated and will appear in the following format:

BasicMathOperation=> add_sub|mul_div

Every line in the file dedicated to specific function type and installed function of that type. This file is automatically updated while installing the function in the agent.

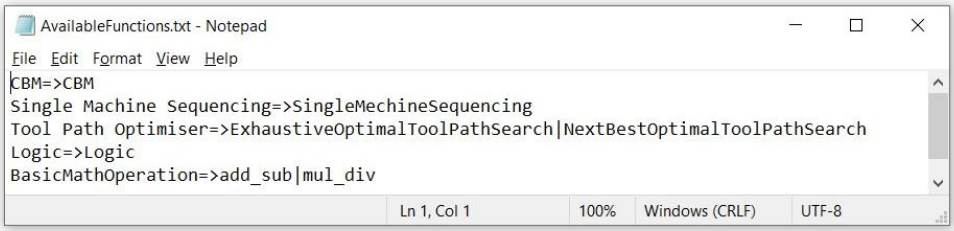


Figure 9.7: Screenshot of AvailableFunctions.txt file

Some functions are dependable on other functions of a specific type. Let's say "Logic" function is dependent on functions with type "CBM", "Single Machine Sequencing", and "BasicMathOperation." These dependencies are maintained by "FunctionDependency.txt" file, and its structure is shown in figure 19 below. This file is automatically updated while installing the function in the agent.

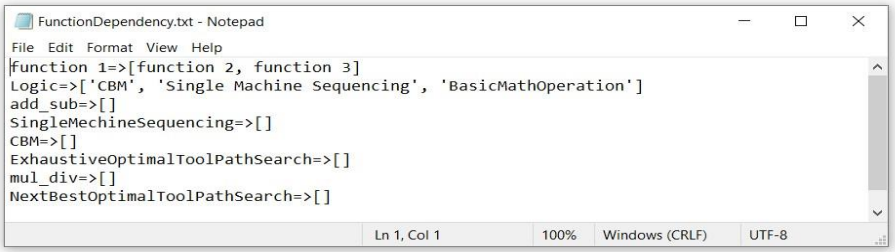


Figure 9.8: Screenshot of FunctionDependency.txt file

As discussed earlier "Logic," function is dependent on functions with type "CBM", "Single Machine Sequencing", and "BasicMathOperation." There could be several functions installed with type "CBM", "Single Machine Sequencing", and "BasicMathOperation" but "Logic" function want to utilise only one function among all installed function of each specific type. Let's say there are two functions are available with function type "BasicMathOperation," i.e., "add_sub" and "mul_div". So which one should "Logic" function utilise? The answer to that question is "FunctionPointingConfig.txt" file. The file points the functions to be utilised for each dependent type of function. The structure of the file is shown in figure 20 below. As seen in figure 20 below, it is pointed that "Logic" function should utilise the "add_sub" function when it need help from any function of type "BasicMathOperation". The standard format used in the file is as follows:

<Function Name>=><type of function>-><dependent function of mentioned type>|<type of function>-><dependent function of mentioned type>

This file is not automatically updated, but it is updated and maintained by the user. This file plays a vital role in the function to function communication, as explained in the subsequent chapters.

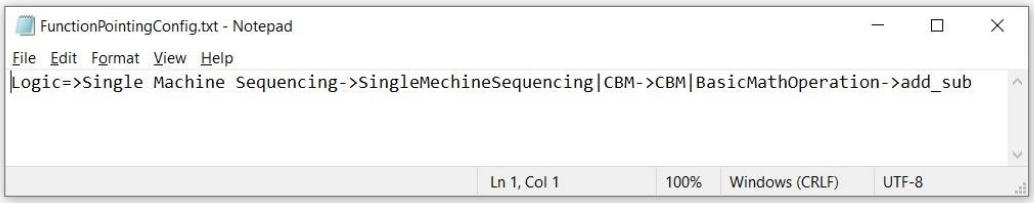


Figure 9.9: Screenshot of FunctionPointingConfig.txt file

Whenever an active function is installed in the agent, it automatically updated in the "ActiveFunctionsList.txt". The file contains the list of active functions present in the agent. Its format is as shown in figure 21 below. The list contains all the active functions from both downloadable and base functions. Let's say the user installs a new active function named "CBM", and then it will appear in the "ActiveFunctionsList.txt" file with the following format:

CBM=>0

Here "0" depicts that while initialisation of the main agent program should not start the CBM function on some thread, which means we don't want to utilise the CBM function for now. User can change “0” to “1”, which can be seen in the file as follows:

CBM=>1

Here "0" depicts that while initialisation of the main agent program should start the CBM function on some thread, which means we want to utilise the CBM function for now. The name of the active function will appear automatically in the file after installation of the active function, but whether to utilise the function or not stays with the user. He/she can change the utilisation mode of function by switching between "0" and "1" in the file for the specific active functions.

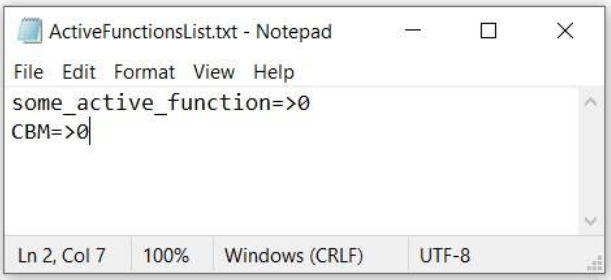


Figure 9.10: Screenshot of ActiveFunctionsList.txt file

Whenever a passive function is installed in the agent, it is updated automatically in the "PassiveFunctionsList.txt". The file contains the list of passive functions present in the agent. Its format is as shown in figure 22 below. The list contains all the passive functions from downloadable functions. Let's say an agent has a "function1" of type "abc" and the user installed a new passive function named "function2" of type "abc", and then it will appear in the "PassiveFunctionsList.txt" file with the following format:

function1=>1

function2=>0

Here "0" in front of "function2" depicts that "function2" is not utilised by base functions when it required to call the function of type "abc". Rather base functions will utilise the "function1" when it needs to call function of type "abc" as there is "1" in front of "function1".

To switch the scenario so that base function can utilise the "function2" rather than "function1" when it required to call function of type "abc"; we have to switch between "0" and "1" for both functions in the file as follows:

function1=>0

function2=>1

The name of the passive function will appear automatically in the file after installing the passive function, but whether that function should be utilised by base functions is decided by the user. He/she

can change the utilisation mode of function by switching between "0" and "1" in the file for the specific active functions.

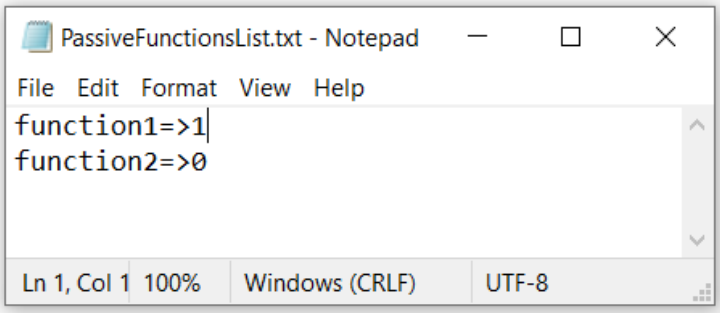


Figure 9.11: Screenshot of PassiveFunctionsList.txt file

It is mandatory that only one function of a specific type should have "1" in front of it. Let's say agent has "function1" and "function2" of type "abc". While it has "function3" and "function4" of type "xyz". Then only one function of each type should have "1" in front of it, leading to one of the situations below:

function1=>1	function1=>0	function1=>1	function1=>0
function2=>0	function2=>1	function2=>0	function2=>1
function3=>1	function3=>1	function3=>0	function3=>0
function4=>0	function4=>0	function4=>1	function4=>1

The "MachineCommunicationConfig.txt" file used to point the downloadable function to the required sensor data files present in the “MachineCommunication” directory. For example, the DAC system is available on the machine that collects live sensor data from the tool spindle. This data will be saved in the “MachineCommunication” directory with a specific data format under a particular name of the file. A tool health monitoring function downloaded in the agent utilises this live sensor data to perform analysis on the same. But how tool health monitoring function will get to know which data file out of all data files available in the “MachineCommunication” directory to utilise. The answer to this question is

"MachineCommunicationConfig.txt" file, the file contains the function name pointing toward the data file name to be utilised from the MachineCommunication directory. As shown in figure 23 with formate followed is as follows:

<function name>=><data file name>

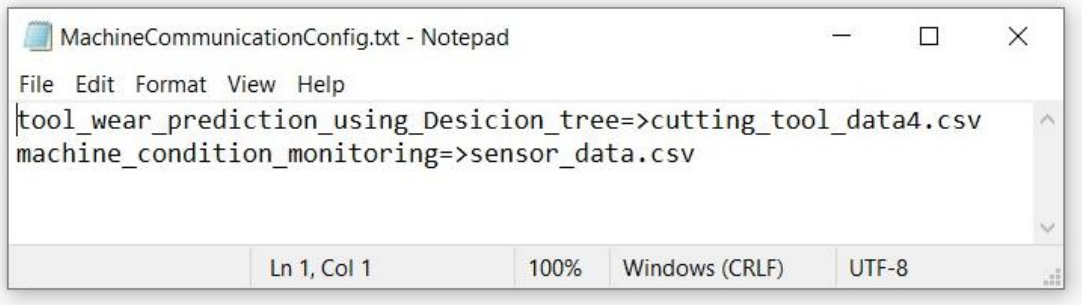


Figure 9.12: Screenshot of MachineCommunicationConfig.txt file

The "IPAddresses.txt" file contains the IP address of the main server along with the port number on the first line and the IP address of the hot standby server along with the port number on the second line. This file needs to be updated by the user before the initialization of the agent.

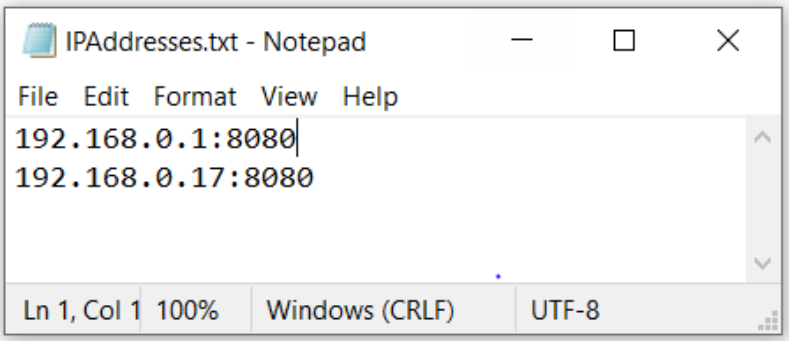


Figure 9.13: Screenshot of IPAddresses.txt file

Chapter 10: System robustness

10.1 heartbeat concept

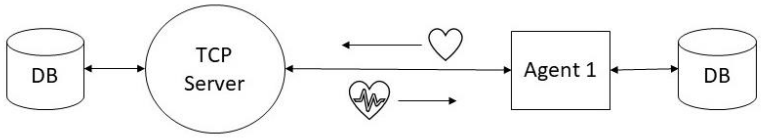


Figure 10.1: Heartbeat schematics

Every 3 seconds, each agent send a heartbeat signal to the server. In response, the server will send its heartbeat signal to the agent. If the server stops receiving the agent's heartbeat signal, it declares that the agent has died with broadcast to other agents.

Similarly, when agents stop a response heartbeat from the server, it gets to know that server is failed. There is also a Heatbeat exchange that happens between the main server and the hot standby server. Here the main server sends a heartbeat to the hot standby server, and in response, the hot standby server sends a heartbeat to the main server. When the main server stops sending the heartbeat to the hot standby server, it assumes that the main server is failed and take communication control in its hand.

The heartbeat is used to introduce the robustness of the system. Heatbeat is used to find if any of the components in the system is dead or alive. If any of the systems is dead, then redundancy in the system is utilised so as to avoid system stoppage or failure.

There could be several failures that could happen in the system. The proposed architecture is robust against the following mentioned failures.

10.2 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 addresses 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.

10.3 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) tasks. 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.

10.4 Standby role agent

A hot standby can be maintained for the sole 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.

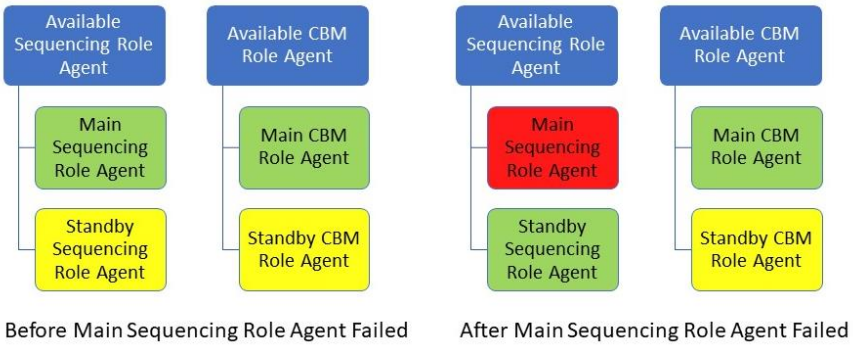


Fig 10.2: robustness against role agent failure

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 11: Case Studies

11.1 Single Machine scheduling on Job Shop

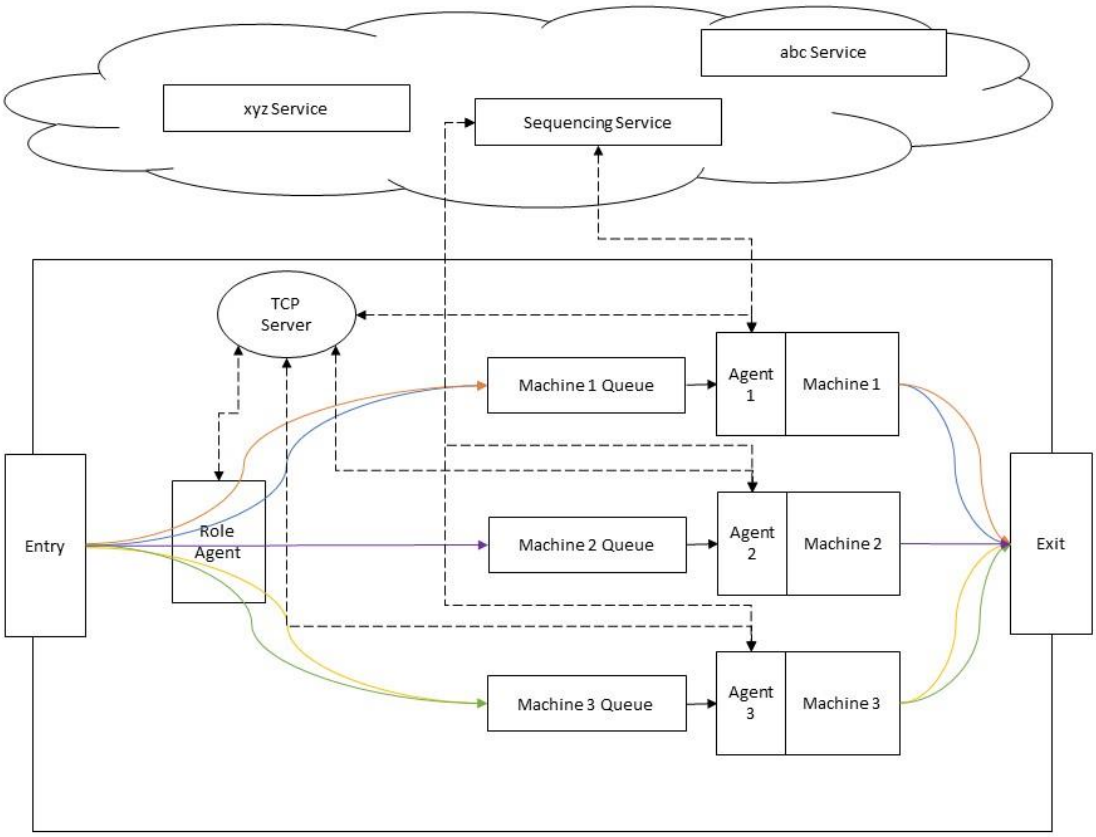


Figure 11.1: Shoop-floor schematics

Consider a simple shop floor with three machines, as shown in figure 30. Job enters from the left side and exits from the right side. Role agent orchestrates the whole sequencing of the shop floor. Also, each machine has an agent enabling the machine with communication capabilities and decision-making capabilities. Role agent has a GUI using which user and new batches to the shop floor. Every batch, once created added to the database of the role agent. Then role agent assigned this batch to one of the machines. The batch is then moved to the queue of that machine. Once the batch leaves the queue, it gets processed on the machine and exits the shop floor. TCP server is utilised for communication between all the agents on the floor.

As explained earlier, the user can configure the generic agent for a specific task. Considering the sequencing and condition monitoring, the author has developed some downloadable functions, as shown in figure 31. These functions include two active functions and two passive functions. “Machine Condition Monitoring” and “Tool Health Monitoring” are active downloadable function types. “Single Machine Sequencing” and “Single Machine Sequencing with Maintenance Consideration” are passive function types.

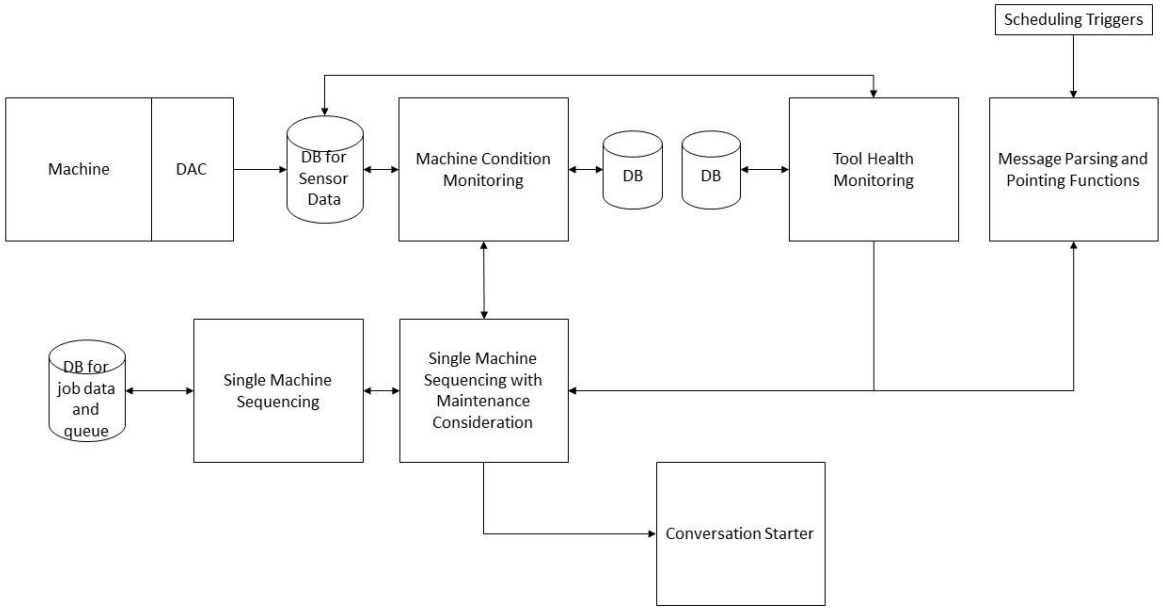


Figure 11.2 Installed function block diagram

“Single Machine Sequencing with Maintenance Consideration” function type has dependencies, and it depends on the “Single Machine Sequencing” and “Machine Condition Monitoring” function type. “Tool Health monitoring” has a dependency on the “Single Machine Sequencing with Maintenance Consideration” function type. One function is developed under each function type by the author for the demonstration purpose. The detail of each function is mentioned in subsequent chapters.

11.1.1 Single Machine Sequencing Function Type

The “Single Machine Sequencing” type function is used to cater to different scenarios of single machine scheduling like penalty calculation in case of new batch arrival, due date, or processing time change of current batch available in the queue. It is a passive function. The function maintains a database of all the

jobs currently present in the queue of the machine. Also, it maintains the optimal job queue as per the jobs available in the queue.

In the case of new batch arrival, the role agent asks the machine agent about the scheduling penalty if the machine agent accepts the batch. That means the machine agent assumes that it has accepted the batch, and it calculates the penalty it will incur as it has added that batch in the queue. It replies to the role agent with this penalty. To support all penalty functionality, the function has a separate internal function that replies to the penalty. The penalty calculation is a generic algorithm followed by all the agents so it can be deployed as a service within the local network or cloud. This service is called by the penalty internal function. Also, penalty calculation can be developed as a separate internal function which is internally called by internal penalty function. Both kind of demonstration as presented with the agent. The scheduling service is explained in subsequent chapters.

When the role agent assigns a new batch to the machine, there is an internal function present which adds this new batch to the queue database and calculates the new optimal sequence of all the batches in the queue.

In case of processing time change or due date change of batch present in the queue of the machine, the reported details are updated by the function in the database. After the update, the optimal sequence of the batches is recalculated using either a separate optimal queue calculation internal function or a generic service. The newly calculated optimal sequence is saved in the database.

This function also acts as a dependent function for other function types. For such cases, it has internal functions that help other functions to read and update the current batch queue of the machine. All the internal functions can be utilized by other functions well through inter function communication.

The base function does not know the internal structure of the function, so they call the execute the internal function with mentioned performative. The execute internal function calls the required internal function based on the mentioned performative and replies.

11.1.1.1 Scheduling Service

These services lie on the internet or local network. In our case, it lies on the internet. The service is in the form of REST API. The machine calls the service and inputs due to date, expected processing time, late penalty and the early penalty for each batch in the queue of the machine. Then service solves a

single machine job scheduling problem which is an NP-Hard problem using the branch and bound method. [26] The scheduling service will take job data in JSON format and reply back the scheduled sequence in the same JSON format. Currently, we are hosting the mentioned service on AWS.

The branch and bound algorithm is explained as follows:

Step 1: Determine the job to be processed last. In any sequence, there is always a job that is processed last. Construct a tree with a node for each possible job to be processed last.

Step 2: Calculate the lower bound on the total penalty (W) associated with the node.

Step 3: Select the node with the lowest lower bound penalty value (parent node) for branching. Each of the remaining jobs can be processed in the next position, so construct nodes for each of these jobs. Develop branches connecting these nodes from the parent node.

Step 4: Calculate the penalty for each of the jobs on the nodes developed in step 3. The penalty is calculated as in step 2 and added to the lower bound of the parent node to obtain the lower bound of the present node.

Step 5: Repeat steps 3 and 4 until the first sequence is determined. This penalty value is the initial minimum penalty value.

Step 6: Search the tree, eliminating all nodes that have lower bounds above the current minimum value.

Step 7: Begin branching on each of the nodes that remain, eliminating nodes where lower bounds are above the current value. Replace the current minimum lower bound with the new minimum value if one is found.

Step 8: Once all other nodes have been eliminated, the current lower bound will be the total penalty for the associated sequence of jobs.

The algorithm returns both penalty and optimal schedule. Penalty function is defined as follows:

if (completion date of batch > due data):

penalty = (completion date of batch - due date) x late penalty

else:

penalty = (due date - completion date of batch) x early penalty

Due to the use of both early and late penalty branch and bound algorithm scheduling leads to just in time. Brach and bound algorithm's time complexity increases exponentially with an increase in the number of batches to schedule. But it has been observed that the number of batches in the queue of any machine are less in number (bellow 15); in that case, it is fine to use branch and bound algorithm as it provides the most optimal batch sequence.

Let there are four batches in the queue. The following table shows batch data in the queue, where Ji, PI, Di, Ei and Li and Unique batch ID, expected processing time, due date, early penalty and a late penalty of the i^{th} batch.

Ji	Pi	Di	Ei	Li
1	1	3	2	3
2	49	176	0	9
3	10	35	3	5
4	27	97	2	3

While calling a scheduling service this table is converted into a JSON format as follows and send to the service:

[{"Ji":1,"Pi":1,"Di":3,"Ei":2,"Li":3},{"Ji":2,"Pi":49,"Di":176,"Ei":0,"Li":9},{"Ji":3,"Pi":10,"Di":35,"Ei":3,"Li":5},{"Ji":4,"Pi":27,"Di":97,"Ei":2,"Li":3}]

In response service send sequence and total penalty in JSON format as follows where array in from of "seq" represent the sequence of batches to be process and number in front of "penalty" represent total penalty which is minimal for given batches:

{"seq":[1,3,2,4],"penalty":96}

11.1.2 Machine Condition Monitoring Function Type

This is an active function, which takes live sensor data as an input, calculates the remaining useful life (RUL) of the machine, then prescribes the maintenance action based on the calculated remaining useful life. It also prescribes the time interval between two consecutive maintenance actions.

The function combines recurrent neural network (RNN) for RUL prediction and reinforcement learning (RL) for maintenance action prescription. There are two parts to it: inference using trained model and weights and continuous learning from the available data. The inference part is developed as a downloadable function. The continuous update can be handled using the development of a digital twin. There could be an internal function that sends live data to the digital twin, which update its weight continuously by learning from the data. After a fixed significant time-interval, the model and weights are transferred to the function. There could be other internal function which infers the model and prescribes the maintenance action based on live sensor data.

11.1.2.1 Architecture of Machine Condition Monitoring Digital Twin

Figure below shows the proposed architecture of the digital twin for the continuous learning process. For training the machine learning algorithm, we need complete data set of the machine from installation to breakdown. At the start, there will be less complete data set available with the digital twin, so the digital twin needs to replicate the data set and learn from it. When new complete data set is available with the digital twin, it should update its model to predict more correctly, which brings the property of continuous adaptation and learning from real-time data.

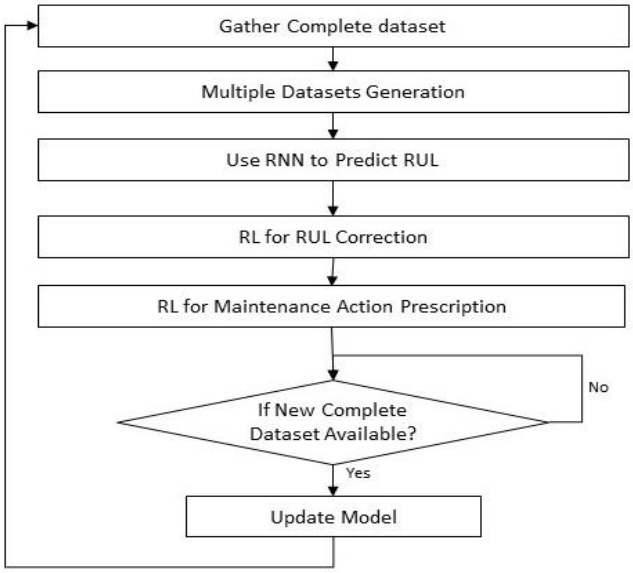


Figure 11.3:2 Digital twin training flowchart and architecture

Let initially, the data available with the digital twin is only four complete datasets [27] in adjoining figure showing Data discretization, the y-axis indicates the health of the machine/component, while the x-axis indicates the operating hours of the machine/component. The data is significantly less to train any deep learning model or reinforcement learning model.

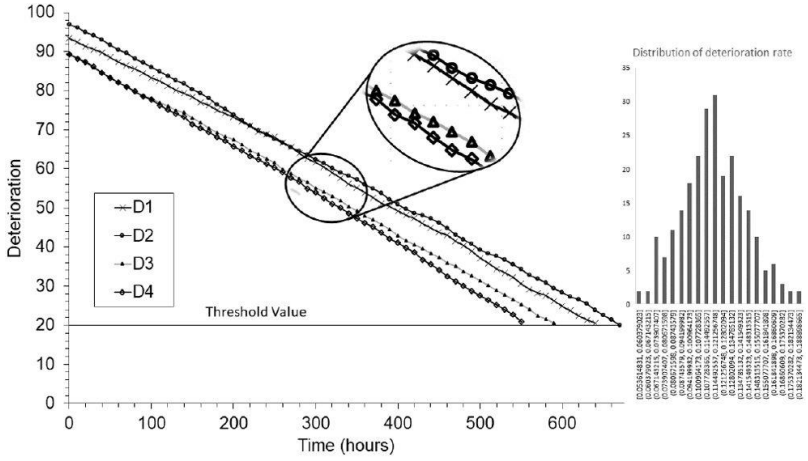


Figure 11.4: Data discretization

As suggested in[27], we can discretize the data into small chunks to calculate the slope of the data. This is will us a distribution of the slops. We can bootstrap the data points of slop from this distribution to generate a new health index vs time profile. Also, if there is a distribution in the initial health index at the

time of installation, we should bootstrap data of the initial health index from its distribution. After the generation of the number of the health index vs time profile, we will have a large number of datasets on which we can train our model. Figure of generated dataset shows large number of the dataset generated by the bootstrapping method. [27]

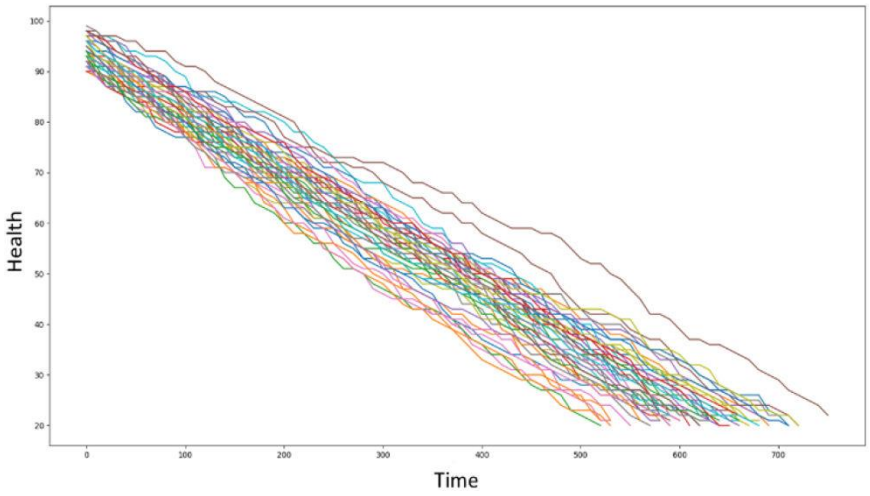


Figure 11.5:3 Generated dataset

Till now, we have talked about the generation of large datasets from a small number of datasets. Also, the dataset contained health index vs time data. But in a real-life scenario, we do not have health index data, rather we have sensor data, and we have to fuse those to get health index data. The data fusion method may vary according to application. Also, another catch is this health index data is nothing but representative of the remaining useful life of the machine/component.

As discussed, we can generate large number of datasets to train an RUL predicting model. The time-series sensor data is fed to the RNN along with the convolution layers model, which predicts the RUL of the machine/component.

The predicted RUL value may differ from the actual RUL value, which can be corrected using the RL algorithm. The RL algorithm penalized when it deviates from the actual RUL value. At the same time, it is rewarded for correctly predicting RUL value [28]. The RUL correction using the RL algorithm is just proposed and not implemented in the project. The predicted RUL is also a health indicator of the machine/component, which is feed to another RL model, which prescribes the maintenance action based on the current health index.

Every data generated by sensors is added to the database of specific machine/components. When the machine goes under breakdown or preventive maintenance, the complete dataset from installation to preventive or breakdown maintenance of the machine is used to update the model. The new data is used to update the parameter distribution, i.e. slop of the health index vs time profile. Then updated distribution is used for bootstrapping the data and generate new datasets. Model in predicting the RUL and prescribing the maintenance action is also updated.

The RUL prediction algorithm is inspired from[29]; in this paper, the combination of 1D convolution layers along with LSMT is presented to predict the RUL of an engine. The authors of the research article call the architecture RUL-net. The RUL-net was primarily developed to deal with the CMAPSS dataset and PHM08 prognostic challenge dataset.

The author has modified the architecture mentioned in this paper to suit the requirement and increase prediction accuracy on the available data set of sensor data. The data set on which the modified version of RUL-net trained was the health index of the gearbox with respect to time. The dataset represents the real scenario where initially we have only seven complete datasets from installation to failure of the different gearbox, as shown below.

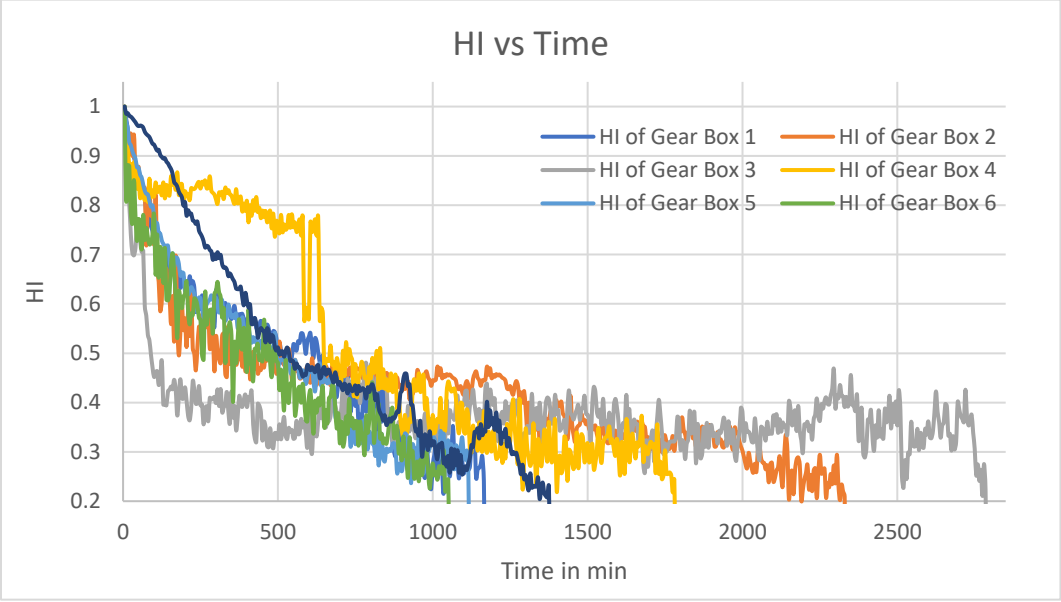


Figure11.6: HI vs Time of gearboxes

```
Model: "sequential_63"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_168 (Conv1D)          (None, 1, 16)             3216
_________________________________________________________________
max_pooling1d_166 (MaxPoolin (None, 1, 16)             0
_________________________________________________________________
conv1d_169 (Conv1D)          (None, 1, 32)             1056
_________________________________________________________________
max_pooling1d_167 (MaxPoolin (None, 1, 32)             0
_________________________________________________________________
conv1d_170 (Conv1D)          (None, 1, 64)             4160
_________________________________________________________________
max_pooling1d_168 (MaxPoolin (None, 1, 64)             0
_________________________________________________________________
bidirectional_115 (Bidirecti (None, 1, 128)            66048
_________________________________________________________________
bidirectional_116 (Bidirecti (None, 1, 128)            98816
_________________________________________________________________
dense_120 (Dense)            (None, 1, 1)              129
=================================================================
Total params: 173,425
Trainable params: 173,425
Non-trainable params: 0
_________________________________________________________________
```

Figure 11.7: Architecture of Deep Learning algorithm to predict RUL

The architecture of the modified RUL-net is shown in figure 36. The network accepts input of 100 sequential data points. Padding is used for initial data points. There are three 1D convolution layers followed by the max-pooling layer. There are 16 filters in the first convolution layer, 32 filters in the second convolution layers and 64 filters in the third convolution layers. Convolution layers help in reducing the numbers of parameters while extracting temporal features. The convolution layers are followed by two layers of bidirectional LSTM layers with 182 units in each layer. The output of bidirectional LSTM is flattened and fed to a fully connected layer that outputs the predicted RUL value.

As there are seven complete datasets, the network was trained on six complete data sets and tested on the remaining one. When the network was trained on a dataset except for the dataset of gearbox one and tested on a dataset of gearbox one, the results of predicted RUL compared with the actual RUL value as shown in above figure.

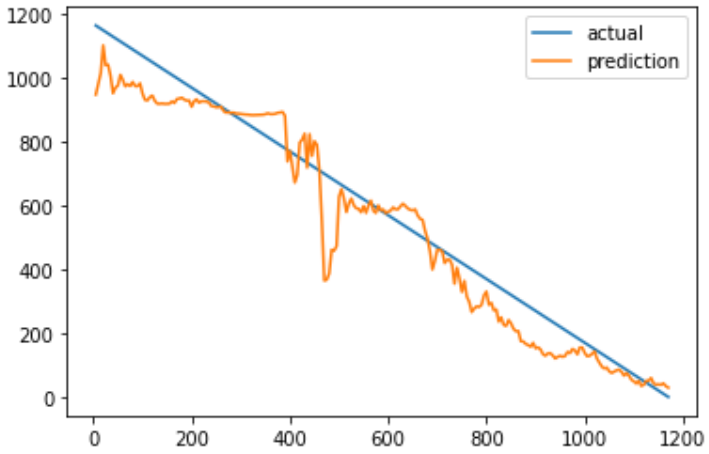


Figure 11.8: RUL predicted vs actual when tested on a dataset of gearbox one

When the network was trained on a dataset except for the dataset of gearbox seven and tested on a dataset of gearbox seven, the results of predicted RUL compared with the actual RUL value is shown in above figure.

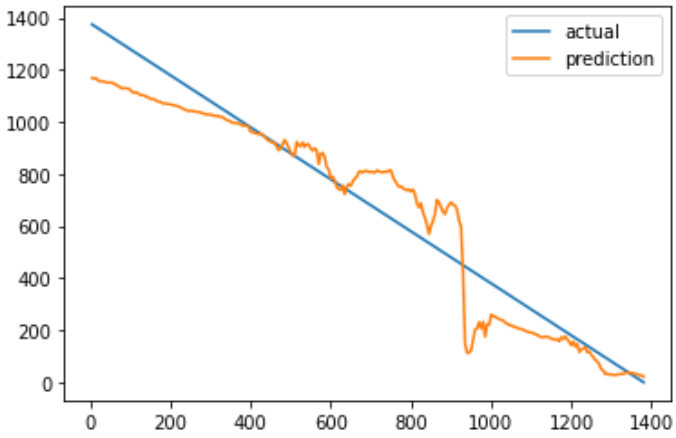


Figure 11.9: RUL predicted vs actual when tested on a dataset of gearbox seven

Once we calculate the RUL value, it is feed to RL algorithm which prescribe the maintenance action along with the interval between two maintenance action. Every maintenance action incurs the cost, and the main aim of the RL algorithm is to reduce the maintenance cost. The cost of maintenance can be reduced by choosing the correct maintenance action at the correct time. The maintenance action can be chosen based on the machine health index, which is the RUL value in this case. The health index of the machine is obtained from the modified RUL-net, as explained earlier. Once we do maintenance of the machine, it

is unnecessary to calculate its RUL value and prescribe the maintenance action at the just time instance. After completing the maintenance action, the user should wait for a specific interval till he/she does the next maintenance action. This specific time interval may vary from industry to industry, considering their maintenance policy. In the current case study, it is assumed that the machine is maintained with 30 days interval between two consecutive maintenance decisions.

There could be a number of maintenance actions available with the machine, i.e. oil change, re-greasing, re-tightening of belts, change of component or even do nothing. Each maintenance action may restore the health of the machine/component or may reduce the rate of decrease in the health of the machine/component. Each maintenance action incurs a different cost. In most cases, breakdown maintenance incurs the highest cost due to its unplanned nature.

In the current case study, due to the unavailability of any machine, we are taking a hardened coating as the component, which wears with time, as an example. Its dimension is the health index in the case study. The health index can be represented by (Markov chain) of 100 states. When the health index of the component reaches 20, it considered as the component is failed and the user replaces the component, and it is called breakdown maintenance. To avoid unplanned breakdown maintenance which incurs a high cost, the industry prefers planned maintenance like minor maintenance and preventive maintenance.

Let's consider component fails due to wears. Due to wear, the thickness of the component decreases. Initially, the component is manufactured with some thickness as no component can be manufactured with the exact specification. So initial thickness is a random variable and follows some distribution. Maximum allowable thickness can be considered as 100 health index, but due to randomness, initial health may be less than 100 within some range. As the component starts to wear, or we can say its health may start decreasing with following some linear relation. Like,

$$H(t) = H(initial) - m \times t$$

Where m is also a random variable and follow some distribution, so, each time wear profile will follow a different path from initial health to final threshold health. To find wear level at the current time, we have to run the RUL prediction algorithm. As the RUL is calculated with 30 days' time interval, the RL algorithm is unaware of wearing a profile in between the span of 30 days.

Maintenance actions taken are No Action (NA), Minor Maintenance (MM), Preventive Maintenance (PM) and Corrective Maintenance (CM/repair). No action is considered as one of the maintenance actions. With No Action, no maintenance action is performed, and the component is allowed to work as it was working before. MM considered as oiling. In this case, it reduces wear rate, but the time for the effect of oil is a random variable. PM is considered as a replacement of that component, but the planned one means a user stop machine with proper scheduling. CM is considered as breakdown maintenance which also includes replacement of component when the health index reaches 20 value. Due to breakdown, the machine stops immediately irrespective of schedules incurring a high cost. CM is a compulsory action required to take when the component fails. Each of these maintenance actions is associated with their corresponding. Cost of CM > Cost of PM > Cost of MM and NA requires no cost as we do nothing.

Reinforcement Learning is a set of algorithms that learn the policy to achieve the desired outcome based on rewards gained from past experience. The learning can either happen through interaction with a real environment or a simulation model that replicates the real environment. Since learning by interaction with a real environment can take an unacceptably long time, the alternative of using a simulation model to replace the real environment is adopted. The approach presented takes the second approach to accelerate the learning process. However, the learning can continue in a real environment once the model is deployed.

The RL algorithm recommends actions based on the current situation and past experience. After performing an action, it observes the outcome of the action in the form of rewards, based on which it updates its policy. After a large number of trials with actions performed at each state or situation, it learns the optimal policy for every state or situation.

An RL algorithm is developed using Q-learning based SMART algorithm with modification for an episodic scenario[27]. Negative rewards are given to the agent as the summation of maintenance cost. Figure 39 shows the Pseudo Code of the modified SMART Algorithm[27]. Here Q state action value is stored for each state-action pair. The decision is made based on the £-greedy algorithm. The algorithm explores with £ probability and exploits with (1-£) probability. After every decision, the Q function is updated, and after every exploited action, average rewards are updated.

1. *Set episode number $n = 0$, and initialize action values $Q_n(x, a) = 0$. Choose the current state x arbitrarily. Set the total reward c_n and total time t_n to zero.*

2. *While $n < MAX\_EPISODES$ do:*

    1. *With high probability p_n, choose an action a that maximizes $Q_n(x, a)$, otherwise choose a random action.*
    2. *Perform action a. Let the state at the next decision epoch be z, the transition time be τ, and r_{imm} be the cumulative reward earned in this epoch as a result of taking action a in state x.*
    3. *Update $Q_n(x, a)$ using:*

$$Q_{n+1}(x, a) \leftarrow (1 - \alpha_n) Q_n(x, a) + \alpha_n \left(r_{imm} - \rho_n \tau + \max_b Q_n(z, b) \right)$$

    4. *In case non-random action was chosen in step 2(a)*

- *Update total reward $c_n \longleftarrow c_n + r_{imm}$*
- *Update total time $t_n \longleftarrow t_n + \tau$*
- *Update average reward $\rho_n \longleftarrow \frac{c_n}{t_n}$*

    5. *Set current state x to new state z, and $n \longleftarrow n + 1$*

Figure 11.10:4 Pseudo Code for modified SMART algorithm

Actions are represented as tuples of Maintenance action and the inspection interval. The inspection interval is 30 days in the mentioned case study. Here we have three inspection actions NA, MM and PM. CM is compulsory action at the breakdown, so we are not considering that in decision-making processes.

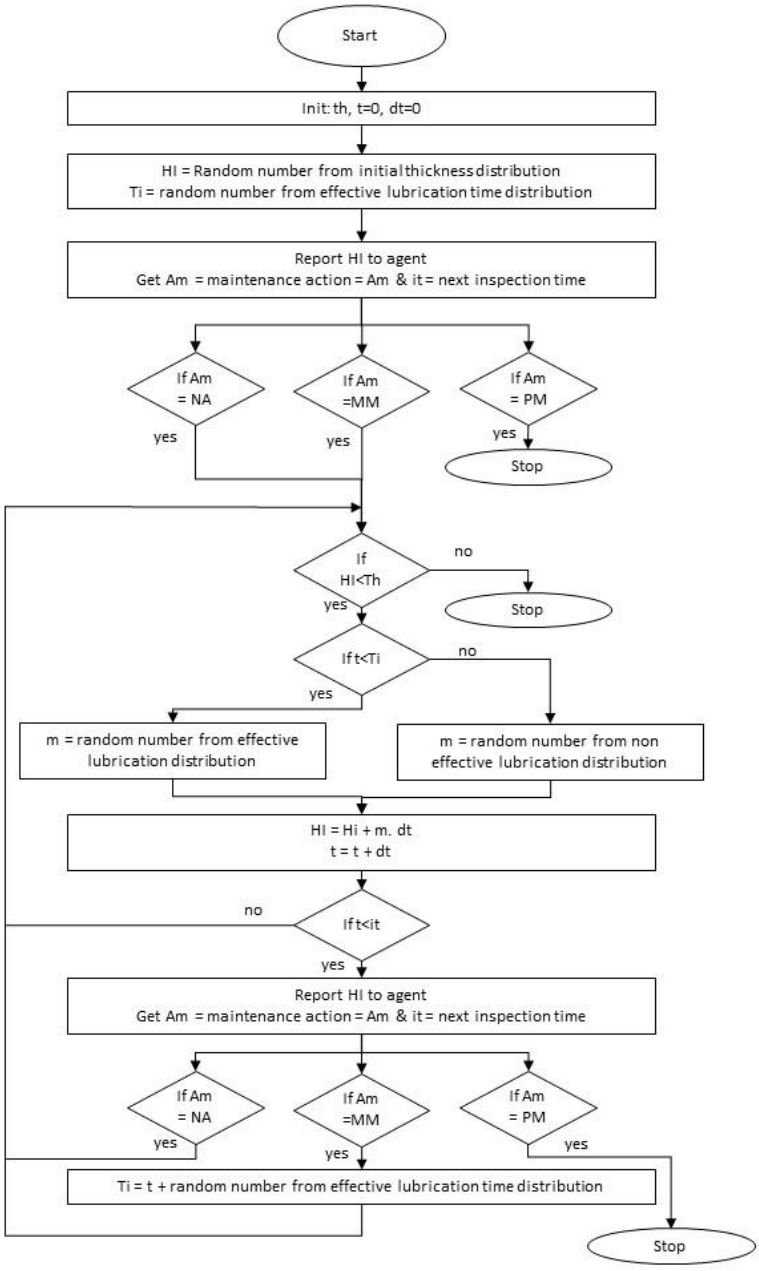


Figure 11.11: Flowchart for environment

There are two parts of the training the RL algorithm; one RL algorithm is which sense current state using inspection and take decision based on optimal policy. The second is an environment that takes the decision of maintenance action and next inspection interval and in return, provide the reward. The environment also helps the agent to sense the current state.

In the environment simulator is shown in figure 11.11, at the start of each replacement cycle (when the component is replaced with the new component), the random number is generated from the initial thickness distribution, which is uniform distribution in our case study. This generated number act as an initial thickness of that new component. It is assumed that oiling is done with replacement to reduce the initial deterioration rate for a specific oiling time. The specific oiling time is also a random variable drawn from the normal distribution. After initialization of each new component deterioration continuous with a linear equation,

$$H(t+1) = H(t) - m \times \Delta t$$

Where m is a random variable drawn from a normal distribution of oiled component, Δt is a small instance of time. Likewise, H(t) is updated for each time instance. After the specific time, the effect of oiling will lapse then the deterioration rate (m) will draw from the other normal distribution of non-oiled component. When MM is performed, the new specific oiling time random variable is drawn, and now deterioration rate (m) is drawn from the oiled component's distribution till specific oiling time lapsed.

11.1.2.2 Machine Condition Monitoring Function

After training both the RUL prediction algorithm and maintenance prescription algorithm, the weights and model can be downloaded and saved into the function's database. The function has an active internal function that read live sensor data from the "MachineCommunication" directory. To know which data file to read from the "MachineCommunication" directory, it uses the "MachineCommunicationConfig.txt", pointing to the active function to correct the data file. An active internal function runs continuously on a thread and produces prescribed output using the model and weights saved in the database. It saves the prescribed maintenance action, time to complete the maintenance and time interval to the database at each prescription cycle. After each prescription cycle, the active internal function goes to sleep as per the prescribed time interval.

There is a callable internal function that other functions can utilize to know the information about the upcoming maintenance schedule. When called, the callable internal function reads the data file of the prescribed maintenance action written by the active internal function and returns the upcoming maintenance action along with its due date and duration.

11.1.3 Tool Health Monitoring Function Type

Each tool has a certain life due to wear. It is a good practice not to use worm tools to avoid quality issues with the produced product and sudden tool breakage, causing unplanned breakdown maintenance and reducing productivity. So, there is a need for a tool wear classification function that classifies the tool's current status based on real-time sensor data.

The “Tool Health Monitoring” Function Type come under the active function category. The DAC of the machine reads live time-series data like vibration, force, acoustic data and process paraments like spindle speed, depth of cut and feed rate. Then DAC extracts the parameters from sensor data and saves live parameters into a file under the “MachineCommunication” directory. The parameters are extracted by the DAC are as follows:

1. Vibration Fundamental Frequency
2. X Vibration Amplitude Fundamental Frequency
3. X Vibration Root Mean Square
4. Y Vibration Amplitude Fundamental Frequency
5. Y Vibration Root Mean Square
6. Z Vibration Amplitude Fundamental Frequency
7. Z Vibration Root Mean Square
8. Force Fundamental Freq
9. X Force Amplitude Fundamental Frequency
10. X Force Root Mean Square
11. Y Force Amplitude Fundamental Frequency
12. Y Force Root Mean Square
13. Sound Fundamental Frequency
14. Sound Amplitude Fundamental Frequency
15. Sound Root Mean Square
16. Spindle Speed
17. Depth of Cut
18. Feed Rate

The function’s active internal function runs on a thread that reads the file updated by the DAC under the “MachineCommuniction” folder. Which file to read from the “MachineCommunication” directory is

mentioned in the “MachineCommunicationConfig.txt” configuration file. The active internal function possesses an inference model, which is saved along with its weights in the function’s database. The active internal function reads the sensor data file every 60 seconds and inputs these parameters to the machine learning model to predict the current state of the tool. The current state can belong to four states labelled as 1, 2, 3 and 4. State 1 belongs to the new tool, while state 4 belong to the worn-out tool. When the tool reaches state 4, it is assumed that the tool has to be changed.

In this hypothetical case study, when the tool reaches state 4 we assume that the machine has failed and we need to change the tool. We consider state 4 as breakdown maintenance with 2 days to repair the machine. When the machine learning algorithm predicts the state 4 of the tool, it calls the internal function named “machine_fail” of “Single Machine Sequencing with Maintenance Consideration” on a separate thread and passes the value time to repair as two days. The “machine_fail” internal function calculates the effect of breakdown on the queue of batches and takes a decision on batch bidding, which is explained in upcoming chapters.

This show that the “Tool Health Monitoring” type function is dependent on the “Single Machine Sequencing with Maintenance Consideration” type function to handle the machine breakdown.

The machine learning model was trained from the data acquired by performing 108 experiments. These 108 experiments were performed with four tools with different level wear, i.e. from new toll to worm tool. The experiments were with each tool by varying cutting parameters, depth of cut, spindle speed and feed rate at three levels. 0.5mm, 1mm and 1.5mm were the depth of cut; 2000, 3000 and 4000rpm were the spindle speed; 80, 100 and 120mm/min were the feed rate chosen for the experiment. A 20mm slot was cut in each experiment on a vertical CNC machine. The data was recorded with the sampling frequency of 800, 5000 and 44100 Hz for vibration data, force data and sound data, respectively. [30] Due to lack of data consistency, only data 90 experiments were considered for machine learning.

Due to the smaller number of datapoints KNN and Decision Tree algorithm was chosen. Classes are fragmented into smaller clusters SVM fails to classify. Hyperparameter tuning was performed to test accuracy. The data set was divided into three equal parts. Two parts were used to train the model, while one part was used to test the model.

11.1.3.1 K Nearest Neighbor Algorithm

With the hyperparameter tuning, it is found that at K = 1, the algorithm test accuracy is highest. The algorithm was able to achieve 100% accuracy on the train data while 80% accuracy in the test data. Following is the confusion matrix for the prediction by the algorithm.

	Wear Level 1	Wear Level 2	Wear Level 3	Wear Level 4
Wear Level 1	4	0	0	3
Wear Level 2	2	3	0	0
Wear Level 3	0	0	3	0
Wear Level 4	1	0	0	14

Figure 11.12 shows the One-vs-Rest ROC. The AUC score achieved was 0.8347.

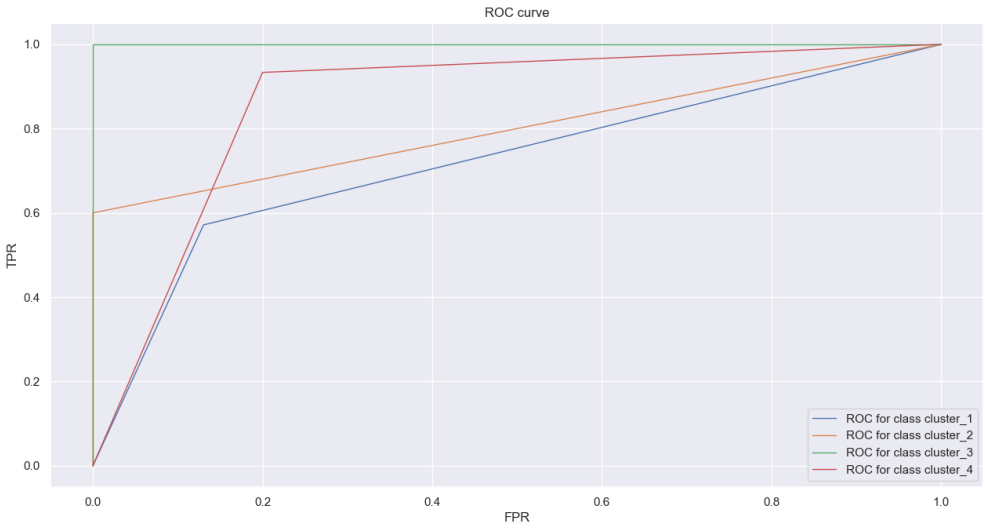


Figure 11.12: ROC for KNN

11.1.3.2 Decision Tree Algorithm

With the hyperparameter tuning, it is found that at a maximum depth of tree as 3 and the minimum number of leaves as 2, the algorithm test accuracy is highest. The algorithm was able to achieve 95% accuracy on the train data while 86.66% accuracy in the test data. Following is the confusion matrix for the prediction by the algorithm.

	Wear Level 1	Wear Level 2	Wear Level 3	Wear Level 4
Wear Level 1	7	0	0	0
Wear Level 2	3	2	0	0
Wear Level 3	0	0	3	0
Wear Level 4	1	0	0	14

Figure 42 shows the One-vs-Rest ROC. The AUC score achieved was 0.913.

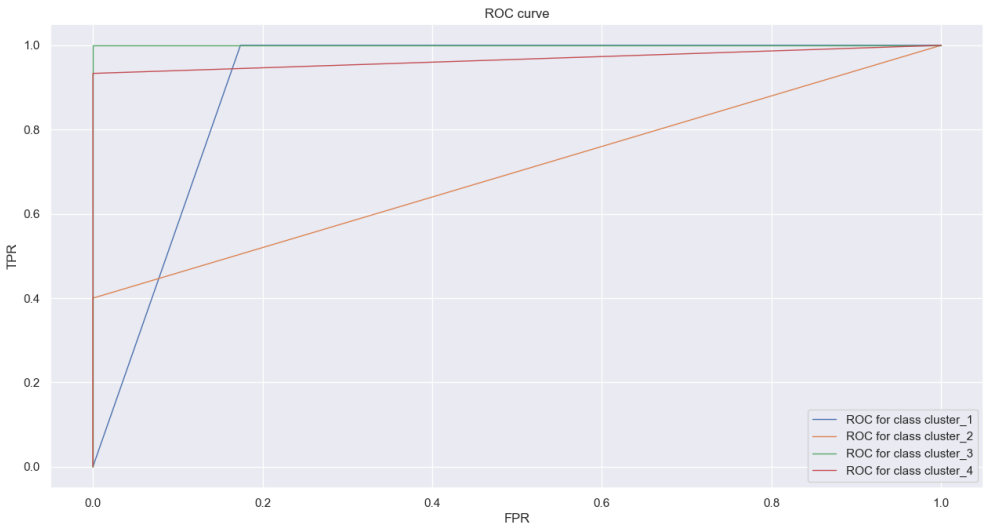


Figure 11.13: ROC for Decision Tree

By comparison, it is clear that the decision tree algorithm seems more accurate on test data, so we choose it to build the "Tool Health Monitoring" function.

11.1.4 Single Machine Sequencing with Maintenance Consideration Function Type

This is a passive function having a dependency on the "Single Machine Sequencing" and "Machine Condition Monitoring" function type. It is meant to handle different scenarios of single machine sequencing considering maintenance actions like new batch arrival, due date, or processing time change of current batch available in the queue and machine failure. This function does not maintain batch queue in its database but utilizes the "Single Machine Sequencing" function type's database by calling its internal function through function-to-function communication. It supports similar communication performative as of "Single Machine Sequencing" type along with some additional to support communication under machine failure scenario.

In agent-to-agent communication regarding the sequencing, the communication happens between the base functions of agents, and then these base functions call the "Machine Condition Monitoring with

Maintenance Consideration" function. The base function does not know the internal structure of the function, so they call the execute the internal function with mentioned performative. The execute internal function calls the required internal function based on the mentioned performative and replies.

Consider the following scenarios for dynamic sequencing on the shop floor:

11.1.4.1 New Batch Arrival on the shop floor

Whenever a new batch arrives on the floor user add that batch using the GUI of the role agent. This initiates the communication on the shop floor to decide which machine will process the newly arrived batch. The role agent will use the contract-net protocol to bid the newly arrived job among all the machines. Role agent will call each machine on the shop floor and ask for a scheduling penalty if the respective machine add a new job to its queue. Based on all penalty replies role agent assign the batch to a machine that offers the lowest penalty.

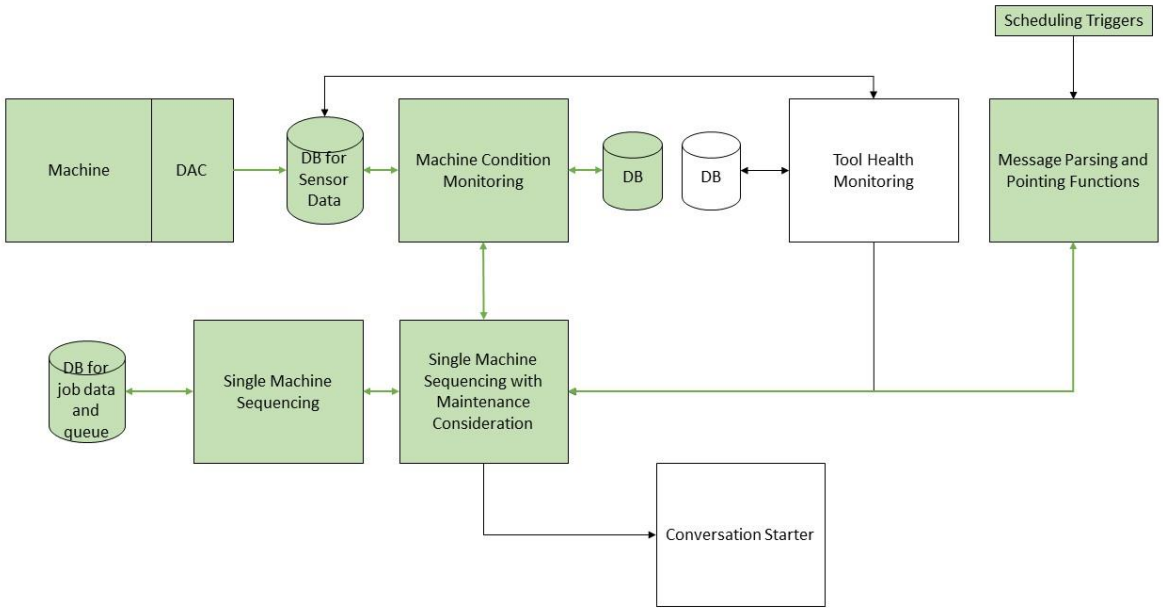


Figure 11.14: Functional block diagram for single machine scheduling for new batch arrival scenario

Let's deep dive into what happens inside the machine agent when it receives the scheduling trigger in the form of communication from the role agent. Among all the downloadable function available in the agent, functions highlighted with green colors are utilized under the new batch arrival scenario. The message

from the role agent is a scheduling trigger. Base libraries parse the message, and using the pointing function; it is passed to function of type “Single Machine Sequencing with Maintenance Consideration”. The decision to which function the message is passed is depend on the performative mentioned in the communication. Let’s say two function support the mentioned performative, them there is a configuration file which helps to decide which function to use. The user maintains this configuration file. In the mentioned case, “Single Machine Sequencing” and “Single Machine Sequencing with Maintenance Consideration” support the same communication performatives. But it is mentioned in the configuration file that the messages should be transferred under the mentioned performatives to “Single Machine Sequencing with Maintenance Consideration” type functions.

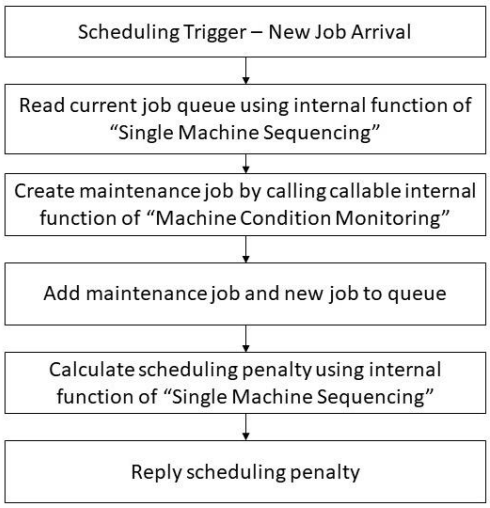


Figure 11.15: Flow diagram for new batch arrival scenario

As directed in the configuration file, the base functions will call the execute the internal function of “Single Machine Sequencing with Maintenance Consideration”. According to performative type, the execute internal function will call the respective internal function, which is the “new_job_arrival_penalty” internal function in our case. The “new_job_arrival_penalty” internal function will call an internal function of “Single Machine Sequencing”, which outputs the current batch queue. Then it calls the callable internal function of “Machine Condition Monitoring” to know details of upcoming maintenance like the type of maintenance scheduled, its duration and due date. It creates a maintenance job using this data and adds to the batch database, which behaves similar to any other batch. The maintenance job will have batch ID, due date, processing time, i.e. maintenance duration, early and a late penalty. The function will also add

a new batch to the queue and then call the penalty calculation internal function of “Single Machine Sequencing” with all of this data. In return, the penalty calculation internal function will give the penalty value it will incur if the machine accepts the batch. The function will return this penalty value to the role agent via base functions.

Communication with contract-net interaction protocol for the case

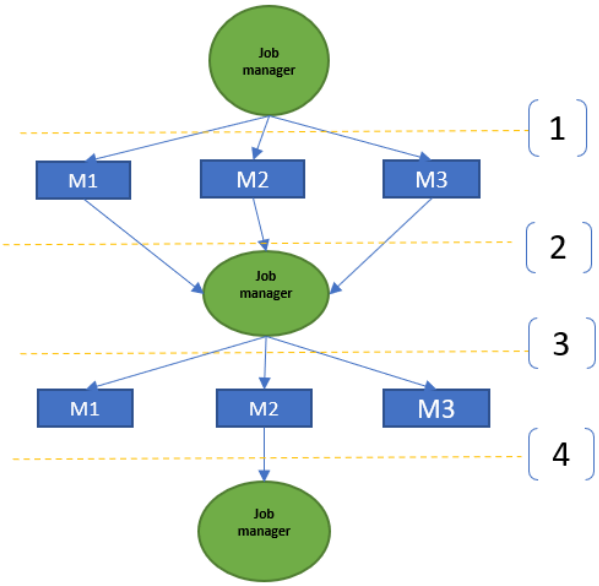


Fig 11.17: conversation flow for case 11.1.4.1

Figure11.17 depicts the flow of conversation among job manager agent and machines M1, M2 and M3 within 4 stages of conversations of contract-net interaction protocol. In the case of new job arrival, when the job manager agent is supposed to use a contract net protocol for auctioning the new job, at stage {1} job manager sends a “call for proposal” performative with job details as shown in the figure 11.18.

```
'{
    'protocol' : 'contract_net_interaction_protocol',
    'performative' : 'call_for_proposal',
    'sender' : 'job_manager',
    'receiver' : 'machine_1, machine_2, machine_3',
    'type' : 'cfp_for_new_job_arrival',
    'content' : '[12,22,07-20-2021,3,2,machine-Agent 2]',
    'conversation_id' : 'CNIP-Cx3f',
}'
```

Fig 11.18: stage 1 message for case 11.1.4.1

At stage {2} of the interaction flow, each of the machine agents propose a cost for processing of the job based on job parameters according to its own passive functions as explained in above section. A typical proposal message is shown below in figure 11.19.

```
'{
    'protocol' : 'contract_net_interaction_protocol',
    'sender' : 'machine_3',
    'receiver' : 'job_manager',
    'performative' : 'propose',
    'type' : 'propose_penalty',
    'content' : '72',
    'conversation_id' : 'CNIP-Cx3f',
}'
```

Fig 11.19: stage 2 message for case 11.1.4.1

After stage {2} is over the manager has to now decide for the best proposer among all the proposals. Lets say that machine agent_2 has proposed to process the job with the least cost, so the job now has to be assigned to machine 2 and all machines are to be informed, so at stage {3} of the interaction flow, the job manager sends an “accept-proposal” performative to machine agent_2 and a “reject proposal” performative to the remaining machine agents. The accept-proposal message is shown in figure 11.20.

```
'{
    'protocol' : 'contract_net_interaction_protocol',
    'sender' : 'job_manager',
    'receiver' : 'machine_2',
    'performative' : 'accept_proposal',
    'type' : 'accept_proposal',
    'content' : 'None',
    'conversation_id' : 'CNIP-Cx3f',
}'
```

Fig 11.20: stage 3 message for case 11.1.4.1

At stage {4} which is a final stage of interaction, the machine agent which has been alloted the job has to notify the job manager whenever the job is done. This is accomplished using an inform performative as shown in figure 11.21.

```
'{
    'protocol' : 'contract_net_interaction_protocol',
    'sender' : 'machine_2',
    'receiver' : 'job_manager',
    'performative' : 'inform',
    'type' : 'inform',
    'content' : 'None',
    'conversation_id' : 'CNIP-Cx3f',
}'
```

Fig 11.21: stage 4 message for case 11.1.4.1

11.1.4.2 Change in Due Date or Processing Time of Batch

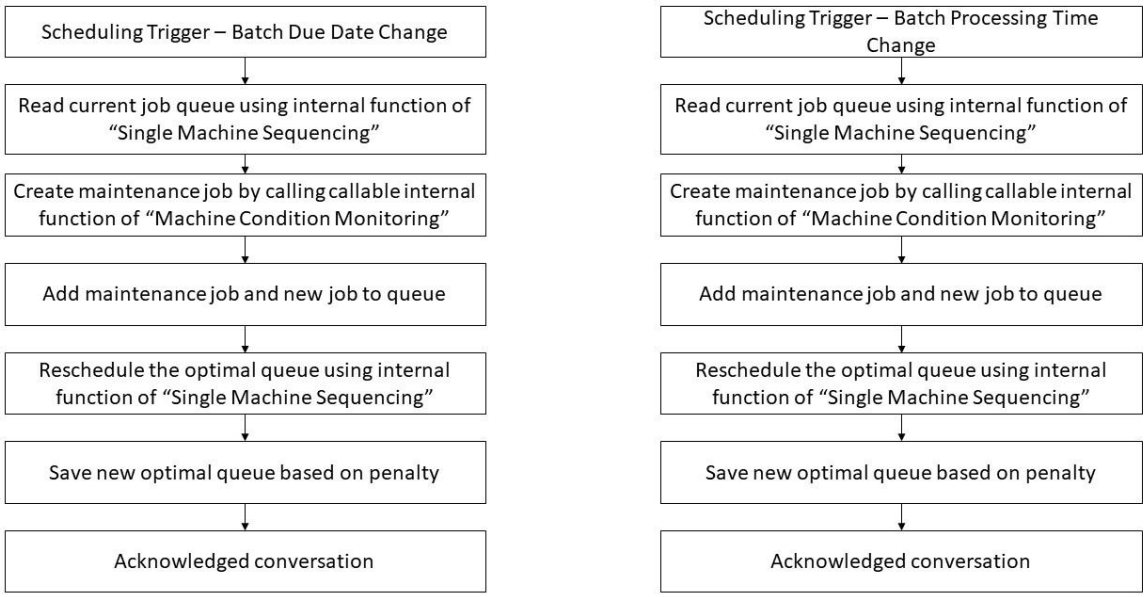
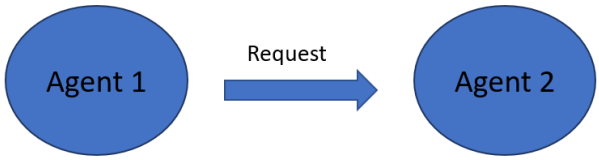


Figure 11.22: Flow diagram for the due date change and processing time change scenario

In case of due date change or processing time change of specific batch present in the queue of the machine, the role agent will pass the changed data using request protocol and specific performative to the machine agent. The message is parsed by base functions at the machine agent and passed the "Single Machine Sequencing with Maintenance Consideration" function's execute internal function. Execute internal function will call the respective internal function based on performative type. In both cases, processing time change or due date change of batch, the respective internal function will call an internal function of "Single Machine Sequencing", which outputs the current batch queue. Then it calls the callable internal function of "Machine Condition Monitoring" to know details of upcoming maintenance like the type of maintenance scheduled, its duration and due date. It creates a maintenance job, as explained earlier. Then it updates the batch's data requested in the message and calls the internal function of "Single Machine Sequencing", which finds the optimal queue for the given batch data. This optimal queue is then sent to another internal function of "Single Machine Sequencing", which updates the optimal queue and saves it in the database.

Communication with request interaction protocol for the case

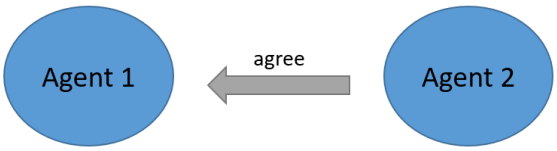


In the above case job manager agent has to ask machine agent to make changes in the job details that has been allotted to it previously. This is accomplished by request interaction protocol. The request for due date change has to be sent with a request performative and a “request_due_date_change” performative type as shown in figure 11.23.

```
'{
    '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:11.23: request message for case 11.1.4.2

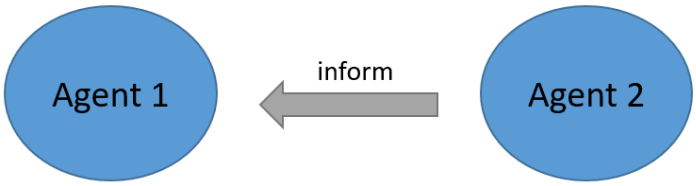
When the agent receives this message, it is supposed to first agree to do the task. The agree message is shown in figure 24.



```
'{
    'protocol' : 'request_interaction_protocol',
    'performative' : 'agree',
    'sender' : 'machine_1',
    'receiver' : 'job_manager',
    'type' : 'agree',
    'content' : 'None',
    'conversation_id' : 'RIP-Cx1e',
}'
```

Fig:11.24: agree message for case 11.1.4.2

After sending an agree message the agent starts performing the task, and after completion of the task the agent informs the job manager that the changes are made.




```
'{
    'protocol' : 'request_interaction_protocol',
    'performative' : 'inform',
    'sender' : 'machine_1',
    'receiver' : 'job_manager',
    'type' : 'inform_add_result',
    'content' : 'success',
    'conversation_id' : 'RIP-Cx1e',
}'
```

Fig:11.25: inform message for case 11.1.4.2

11.1.4.3 Machine Failure

All functions that play a role in the re-scheduling process when one of the machines fails are highlighted in pink. “Tool Health Monitoring” type function is an active function that continuously monitors the health state of the tool, as explained earlier. Whenever the tool’s health state reaches state 4, the active internal function of “Tool Health Monitoring” calls the “machine_fail” internal function of “Single Machine Sequencing with Maintenance Consideration” on the separate thread and passes time to repair data.

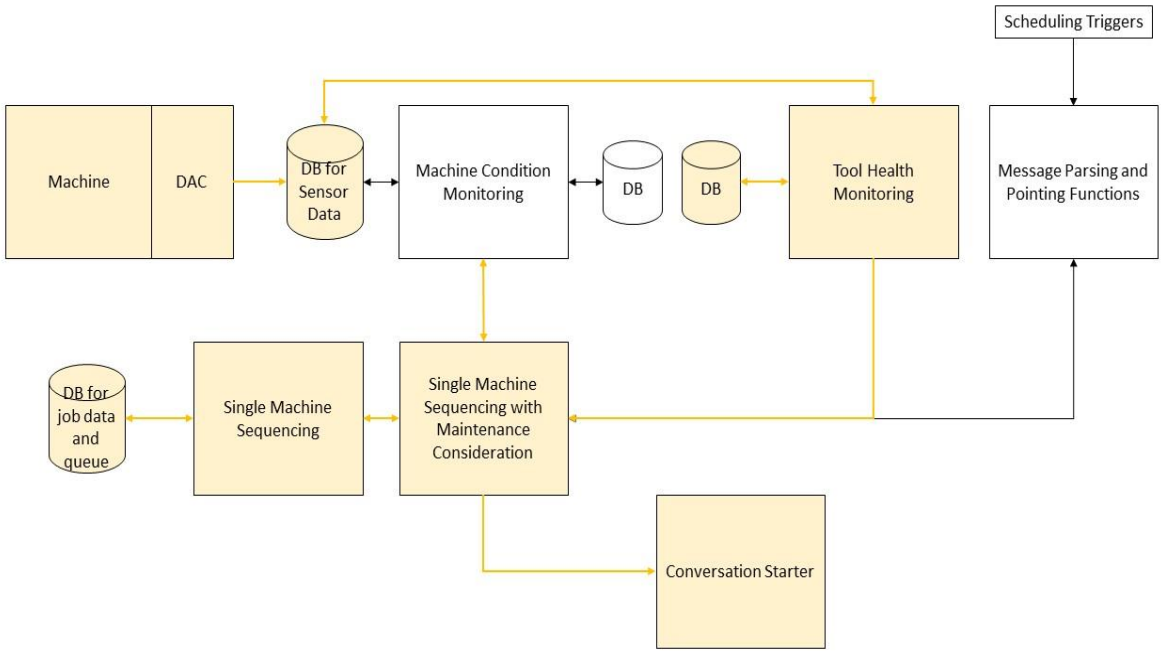


Figure 11.26: Functional block diagram for single machine scheduling for the machine failure scenario

The “machine_fail” internal function will call an internal function of “Single Machine Sequencing”, which outputs the current batch queue. As per time to failure data received from the active internal function of “Tool Health Monitoring”, it updated the due date of all the batches. Let’s say the batch has a due date in 10 days and the time to repair is 2 days; then the updated due date will be 8 days. The “Single Machine Sequencing with Maintenance Consideration” type function maintain a file in its database named “penalty_threshold.txt”, this file contains the threshold penalty value, which is user-defined. After updating all the due dates of batches, the internal function reads this file to know the threshold penalty value. Then it calls the internal function of the “Single Machine Sequencing” type function, which calculates the penalty using all the batches in the queue with updated due date. If the penalty calculated is above the threshold penalty, then it removes the first batch from the optimal queue and recalculates the penalty value with the new queue of batches. This process is repeated till the calculated penalty value is less than the threshold penalty value. Every time this process is repeated, a batch is removed from the queue. After completion of removal process batches, removed batches are bided to other machines by starting a conversation for each removed batch using the conversation starter function and contract-net protocol.

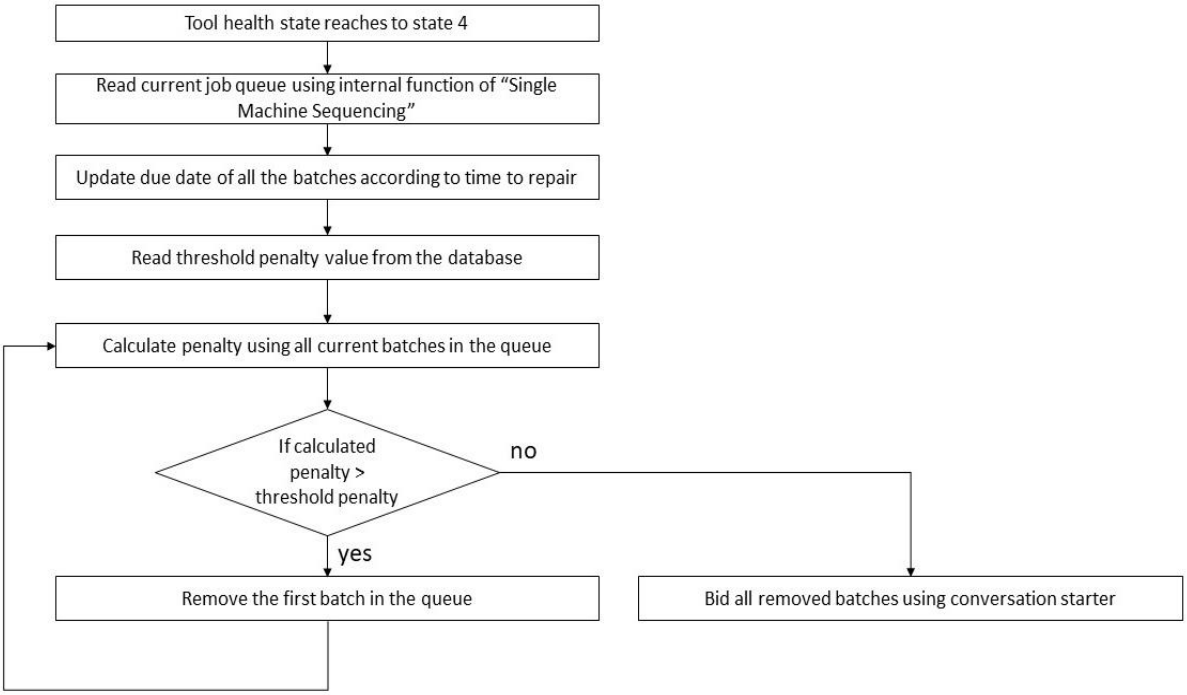


Figure 11.27: Flow diagram for the machine failure scenario

11.2 Tool Path Optimization

Each batch containing jobs will have different geometry. There is a need for different CNC code for each batch. Because of dynamic scheduling, we don't know which machine will end up processing the batch. Under such condition, it is recommended that the CNC code should only be loaded once the batch reaches the end of the machine's queue. Some mechanism is required so that machine can sense which batch it will process next and find the job's CAD file or CNC code. In our case, we are building capability into a CNC machine to generate the CAM file based on the CAD file of the job.

According to the proposed way, the batch can have a RFID tag or BAR code on it, which the machine can read using RFID reader or BAR code reader. The RFID tag or BAR code will contain a link to a CAD file, which could be stored in the cloud or local server. The machine will download the CAD file using this link. Let's say we are using RFID reader, which is hardware present with the machine and comes under physical space. The firmware in the RFID reader will download the CAD file and keep that in the "MachineCommunication" directory.

Whenever a worker on the machine tries to load the next batch on the machine, he/she will use the “Load Next Batch” button on the machine GUI, which will call the “Tool Path Optimizer” function in the background. The “Tool Path Optimizer” type function is a passive function. It reads the CAD file present in the “MachineCommuniaction” directory and creates a CAM file which is a G-Code file. It saves it in the database as well as returns in G-code in the string format. This G-Code file can feed to the machine to process the jobs in the batch. The process flow is depicted in figure 48. The block in the orange is the “Tool Path Optimizer” type function.

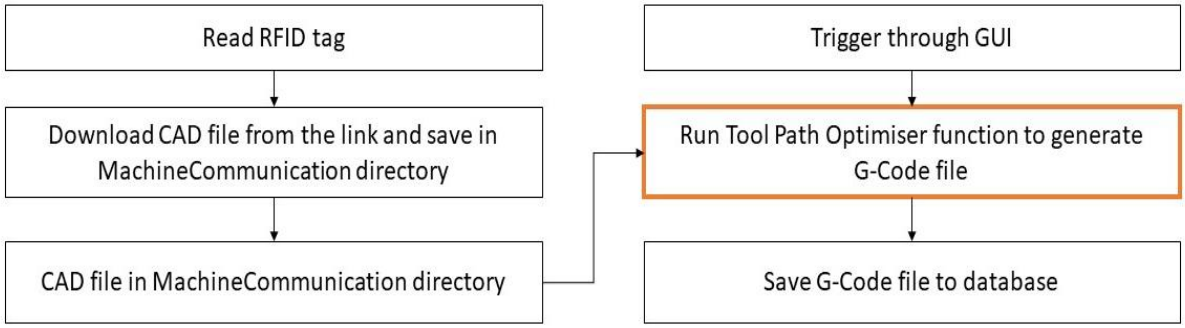


Figure 11.28: Flow diagram for tool path optimisation

The user can download different functions of the “Tool Path Optimizer” type and use one of them as per his/her requirement. This case study showcases that users can download and install multiple functions of the same type and use one of them by configuring the default function in the “FunctionPointingConfig.txt” file. Let’s say there a GUI, which is also a downloadable function that triggers the “Tool Path Optimizer” type function when the user clicks the specific button. GUI also passes all the required information like tool radius, feed rate, spindle speed and the base plane of the job. There are multiple functions available, so the “FunctionPointingConfig.txt” file helps decide the GUI which function to call among all the installed functions. We have developed the following functions of the “Tool Path Optimizer” type for demonstration purpose. These functions are of the same type, so the data input and output format of these function are similar, although the tool path generation strategy may differ. Any of the following functions can be used in place of the orange block of the figure 48 by configuring it in the “FunctionPointingConfig.txt” file. We are also assuming the CNC machine to be a simple PCB drilling machine that drills multiple holes on the PCB. So, the tool path optimizer has the task to select a sequence of holes for drilling that provides the least distance cost. This is a replication of the travelling salesman problem.

11.2.1 Function with Brute Force Method

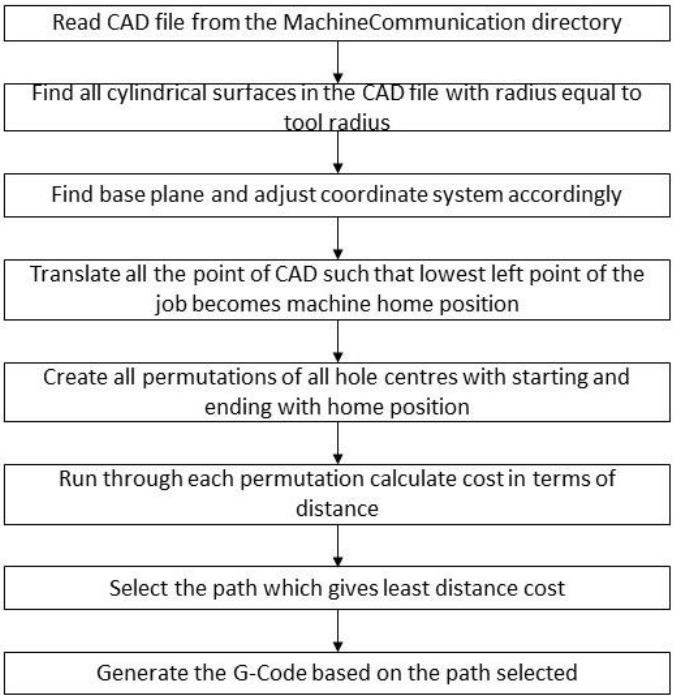


Figure 11.29: Flowchart for brute force method tool path optimiser

The function reads a CAD file of step file format from the “MachineCommunication” directory and finds all the cylindrical surfaces from the file having a radius equal to the tool radius. Then it finds the base plane of the CAD file and rotates the CAD according to the machine’s base plane. It translates all the points in the CAD so that the lowest left point matches the home position of the machine.

It creates all the permutation of the hole sequences with starting and endpoint as the home position. Then it evaluates each permutation and selects that permutation that offers the least tool travel distance. Using this sequence of holes generates the G-Code and return to the calling function in the form of a string output. Also, it saves the code in its database in txt format.

11.2.2 Function with Next Best Method

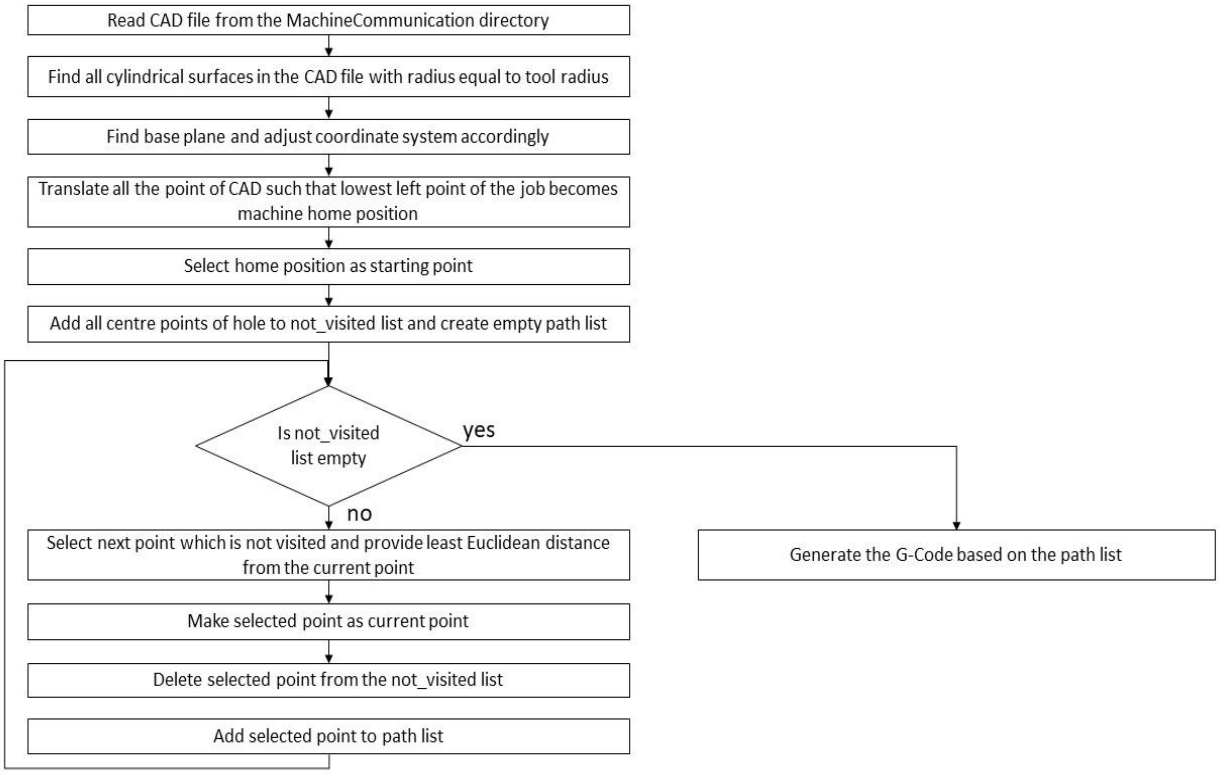


Figure 11.30: Flowchart for next best method tool path optimiser

The function reads a CAD file of step file format from the “MachineCommunication” directory and finds all the cylindrical surfaces from the file having a radius equal to the tool radius. Then it finds the base plane of the CAD file and rotates the CAD according to the machine’s base plane. It translates all the points in the CAD so that the lowest left point matches the home position of the machine.

It creates a list of all the hole coordinates named “not visited” and an empty list named “path”. It is assumed that the tool will be at the home position at the start of processing. It makes the home position the current position or starting position. Then it selects the next point based on the point which offers the least Euclidian distance between the current point and the selected point. It makes the selected point a current point while adding the selected point to the “path” list and deleting the selected point from the “not_visited” list. Repeat the next point selection process till the “not_visited” list become empty. Using this sequence of holes stored in the “path” list generates the G-Code and return to the calling function in the form of a string output. Also, it saves the code in its database in txt format.

Chapter 12: 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.

12.1 PADE (python agent development platform) (https://github.com/grei-ufc/pade)

The python agent development platform is an open-source project under MIT license terms and is developed by Smart Grids Group (GREI) in the 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 the 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.

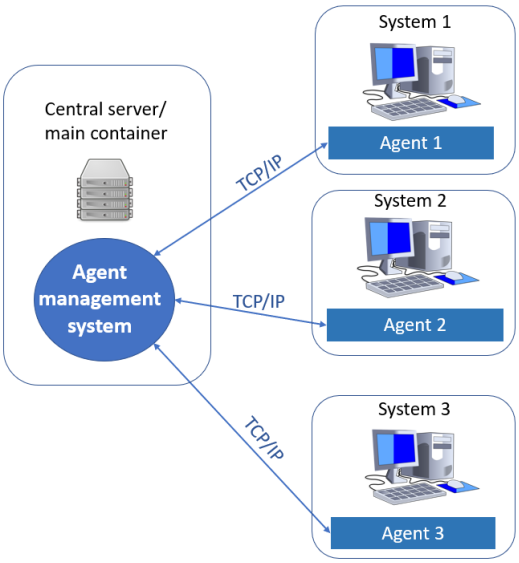


Fig 12.1: architecture for PADE agents

Each platform has a unique service which is named an 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 the 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.

12.2 Drawbacks and challenges while implementation of PADE:

1.PADE is 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 initiating an AMS server, with agents on a single platform. These agents can 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 used interaction protocols modules. But it lacks in explaining how does this works 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 the network: It was observed that PADE agents have a very low response time to initiate and start running, this cannot be said when they are terminated tough. 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 the state of the agent and freezes the agent activities on the platform. Therefore, it does not seem fit for a 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 is. This specifically creates problems when the software is used at a large scale, when an 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 the above-mentioned drawbacks.

12.3 Smart python agent development environment (SPADE) (https://github.com/javipalanca/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 an 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.

12.4 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 that 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 Underdevelopment: 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 an 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 is. This specifically creates problems when the software is used at a large scale, when an agent may have to handle a lot of messages at the same time.

Chapter 13: 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 a fixed and open set of guidelines to judge a smart product. The 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.

13.1 What has been achieved on the server-side:

- Initial protocols (at server-side) for agent-server interactions during agent initiation in a network are defined.
- Mediating one to one and one to many interactions.
- Broadcasting to all agents as well as forwarding message to particular target agents is 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 situations all the agents are informed of this activity.
- Stable operation of the server even when some agent in the network fails
- Handling multiple agent messages at the same time.

13.2 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.
- An agent can make an 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).
- The 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 a certain time to gather replies from all the agents while not hampering its other operations.
- The agent has a GUI that shows interactions conversation logs with other agents.
- The agent can recognize and distinguish one conversation from another conversation with the help of conversation ids and thus it can 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.

Chapter 14: Future scope

The multi-agent system proposed and explained in this piece of work is 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.

14.1 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.

14.2 On the agent side

- As additional functionalities get added to the system, hence there would be a need for an Agent GUI for the operators, this may include a showcase of conversation logs, a 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 therefore the coding for those agents is to be done based on a 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.

14.3 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 the 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 the 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.

14.4 Agent management

In the agent management section, there will be an “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 the 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.

14.5 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 the screen. After the installation, the application restarts, and the agent starts 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 accesses a defined path where all the package scripts that are shared by the server are stored and can be accessed.

Demonstrate the implementation of ZVIE's criteria to transform the conventional CNC machine into a smart CNC machine while implementing all case studies mentioned in chapter 5. While implementing the criteria, there has to be standardization of communication protocol, format and semantics to achieve interoperability.

The scheduling service mentioned in the case study can be improved in terms of the time required to run the algorithm with the help of heuristics. Development of digital twin of shop floor which is connected to each machine and ERP system. The digital twin may contain the information of all the batches on the shop floor and information of all machine, including their capabilities, health status and machine queue. The digital twin can tweak the scheduling of the shop floor dynamically to absorb the unplanned events that happened on the shop floor.

There is scope in changing the strategy of RUL prediction as a piecewise linear RUL function as represented in[29]. Deploy the digital twin for CBM as proposed in the case study, which accepts the sensor data to predict the RUL of component/machine and prescribes the maintenance action and inspection interval for long term cost optimization. Currently, all the AI modules of the proposed digital twin are ready, but they need to be integrated with an actual physical machine. Also, the way of communication of sensor data to digital twin and maintenance action prescription from asset need to be finalized. Also, rather than a bootstrapping method for GAN can be explored for data generation.

Another digital twin can be developed, which is focused on CAD visualization. It could have the capability to augment live and historical sensor data on the CAD. Along with this, it should able to respond against any request to present the specification of an asset, its maintenance details, manual and standard operating procedure. It could have the capability to communicate with the human in natural language so that human can query the digital twin in natural language. At the same time, it should have M2M communication capabilities for sensor data collection and communication with other assets or digital twins. The query could be for past data or future and current predictions using machine learning algorithms.

Actual deployment of RFID reader with the machine and RFID tag with batches to showcase the tool optimization case study as mentioned in previous sections. Also, system robustness is only proposed but not actually implemented either in software or hardware.

Chapter 15: Resources

GitHub repository for MAS software:

https://github.com/IITB-MAS/version-1

Link to all the data and Codes:

https://drive.google.com/drive/folders/1FPrnra_B1YJzFQq5sysRhVTSY5MrZFT5?usp=sharing

Publication:

Journal 1) Adsule, A., Kulkarni, M. and Tewari, A. (2020), Reinforcement learning for optimal policy learning in condition-based maintenance. IET Collab. Intell. Manuf., 2: 182-188. https://doi.org/10.1049/iet-cim.2020.0022

Chapter 16: 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] 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.

[10] J. Dong, S. Chen, and J. J. Jeng, “Event-based Blackboard architecture for multi-agent systems,” *Int. Conf. Inf. Technol. Coding Comput. ITCC*, vol. 2, pp. 379–384, 2005, doi: 10.1109/itcc.2005.149.

[11] M. L. Nunes, A. C. Pereira, and A. C. Alves, “Smart products development approaches for Industry 4.0,” *Procedia Manuf.*, vol. 13, pp. 1215–1222, 2017, doi: 10.1016/j.promfg.2017.09.035.

[12] M. Kasunic, “Measuring Systems Interoperability: Challenges and opportunities.,” *Carnegie-Mellon Univ Pittsburgh Pa Softw. Eng. Inst*, 2001.

[13] 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.

[14] R. Evans *et al.*, “MESSAGE: Methodology for engineering systems of software agents,” *Eurescom, Edin*, no. September 2001, pp. 223–907, 2001.

[15] H. Tang, D. Li, S. Wang, and Z. Dong, “CASOA: An Architecture for Agent-Based Manufacturing System in the Context of Industry 4.0,” *IEEE Access*, vol. 6, pp. 12746–12754, 2017, doi: 10.1109/ACCESS.2017.2758160.

[16] X. Zheng, F. Psarommatis, P. Petrali, C. Turrin, J. Lu, and D. Kiritsis, “A quality-oriented digital twin modelling method for manufacturing processes based on a multi-agent architecture,” *Procedia Manuf.*, vol. 51, no. November 2020, pp. 309–315, 2020, doi: 10.1016/j.promfg.2020.10.044.

[17] Y. Liu, L. Wang, Y. Wang, X. V. Wang, and L. Zhang, “Multi-agent-based scheduling in cloud manufacturing with dynamic task arrivals,” *Procedia CIRP*, vol. 72, pp. 953–960, 2018, doi: 10.1016/j.procir.2018.03.138.

[18] B. Afsar, D. Podkopaev, and K. Miettinen, “Data-driven Interactive Multiobjective Optimization: Challenges and a Generic Multi-agent Architecture,” *Procedia Comput. Sci.*, vol. 176, pp. 281–290, 2020, doi: 10.1016/j.procs.2020.08.030.

[19] K. A. Group, T. Finin, D. Mckay, and R. Fritzson, *An Overview of KQML: A Knowledge Query and Manipulation Language*, no. July. 1992.

[20] FIPA, “Index @ Www.Fipa.Org.” [Online]. Available: http://www.fipa.org/.

[21] Foundation for Intelligent Physical Agent (FIPA), “FIPA Request Interation Protocol Specification,” 2002.

[22] (Fundation 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.

[23] F. For and I. Physical, "FIPA Subscribe Interaction Protocol Specification," *Architecture*, 2002.

[24] Foundation for intelligent physical agents, "The foundation for intelligent physical agents," *IEEE Comput. Soceity*, p. 1, 2013.

[25] FIPA, "FIPA Propose Interaction Protocol Specification," *IEEE Comput. Soceity*, p. 1, 2013.

[26] D. R. Sule, "Production Planning and Industrial Scheduling," *Production Planning and Industrial Scheduling*. 2007, doi: 10.1201/9781420044218.

[27] A. Adsule, M. Kulkarni, and A. Tewari, "Reinforcement learning for optimal policy learning in condition-based maintenance," *IET Collab. Intell. Manuf.*, vol. 2, no. 4, pp. 182–188, 2020, doi: 10.1049/IET-CIM.2020.0022.

[28] D. Kozjek, A. Malus, and R. Vrabič, "Multi-objective adjustment of remaining useful life predictions based on reinforcement learning," *Procedia CIRP*, vol. 93, pp. 425–430, 2020, doi: 10.1016/j.procir.2020.03.051.

[29] L. Jayasinghe, T. Samarasinghe, C. Yuenv, J. C. Ni Low, and S. Sam Ge, "Temporal convolutional memory networks for remaining useful life estimation of industrial machinery," *Proc. IEEE Int. Conf. Ind. Technol.*, vol. 2019-February, pp. 915–920, 2019, doi: 10.1109/ICIT.2019.8754956.

[30] "Data Analytics in Tool Condition Monitoring," 2017.