



PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA  
Ministry of Higher Education and Scientific Research  
University of Mohamed Khider – BISKRA  
Faculty of Exact Sciences, Science of Nature and Life

## Computer Science Department

Ordre N° : IA8/M2/2021

### Thesis

Submitted in fulfilment of the requirements for the Masters degree in

## Computer science

Option : Artificial Intelligence

---

# A combinatorial auction-based approach for IoT Service composition

---

By :  
**NECIB HANA**

Members of the jury :

BEN SGHIR NADIA	MCA	President
MERIZIG Abdelhak	MCB	Supervisor
BENDAHMANE Toufik	MAA	Member

**Session 2021**



## Acknowledgements

First and foremost, praises and thanks to the God, the Almighty, for his showers of blessings throughout my research work to complete it successfully.

I would like to express my deep and sincere gratitude to my research supervisor **Dr. Abdelhak Merizig**, for giving me the opportunity and providing invaluable guidance throughout this work. His dynamism, vision, sincerity and motivation have deeply inspired me. It was a great privilege and honor to work under his guidance. I am grateful for his support and understanding all along the way. I extend my appreciation to all teachers of the **Computer Science Department** who helped me in my education.

I am extremely grateful to **my parents** for their continuous and unparalleled love, prayers, caring and sacrifices for educating and preparing me for my future. Also for standing with me against all the obstacles i faced. I am forever indebted to them for giving me the opportunities and experiences that have made me who I am. This journey would not have been possible if not for them, and I dedicate this milestone to them and to my dear grandmother's soul.

My special thanks goes to my sisters **Nassima, Kheira** and **Sana**, to my brothers **Amine** and **Said** for being always by myside. Many thanks to all of them for their valuable support i am grateful for what they have offred me. I would like to thank also my nephews and nieces for their love.

Last but not least, i am extending my heartfelt thanks to my friends and to my beloved ones especially **Aymen** and **Selma** for the keen interest shown to complete this project successfully.

Necib Hana

## ملخص :

في الآونة الأخيرة ، أحدثت إنترنت الأشياء (Internet of Things) تحولاً في مجال الزراعة ، مما مكن المزارعين من مواجهة التحديات الرئيسية في حياتهم اليومية. تحاول الاختراعات الجديدة في تطبيق إنترنت الأشياء حل المشكلات عن طريق تحسين معايير الإنتاج مثل جودة الزراعة وكميتها واستدامتها. يؤثر المزارعون في هذا المجال لمراقبة نمو المحاصيل ورطوبة التربة وما إلى ذلك. بالإضافة إلى ذلك ، يمكن استخدام أجهزة الاستشعار للتحكم عن بعد في معدات الري. كما أنهم يتخذون قرارات بسهولة بشأن إمدادات مياه المزرعة ، مع مراعاة المعايير الوظيفية لتقليل استهلاك المياه وسرعة الاستجابة لطلبات المستخدمين. في الواقع ، لضمان نظام الري (irrigation system) والاستجابة لكل من المعلمات الوظيفية وغير الوظيفية الممثلة في جودة الخدمة (QoS) ، يحتاج العملاء إلى إنشاء تركيبة الخدمة (service composition). حالياً ، يعد تكوين الخدمة أحد المشكلات الرئيسية في إنترنت الأشياء نظراً لتوافر الخدمات الذرية التي يمكن أن تخدم وظائف مماثلة مناسبة. هناك مجموعة من المرشحين لكل خدمة ذرية في تركيب الخدمات و يتم اختيار خدمة مرشح مناسبة منها بناءً على معايير معينة. جودة الخدمة (QoS) هي أحد معايير اختيار الخدمات المناسبة. يحول المشروع الحالي مشكلة تكوين الخدمة إلى مشكلة تحسين (optimization method). علاوة على ذلك ، في هذا المشروع ، نقترح أيضاً نهج المزاد التوافقي (combinatorial auctions) (طريقة العطاءات) من أجل تقديم أفضل اختيار للخدمات لإنشاء تكوين الخدمة.

**الكلمات المفتاحية :** نظام الري ، تكوين الخدمة ، الخدمات الذرية ، جودة الخدمة ، إنترنت الأشياء ، المزادات التوافقية ، طريقة التحسين ، -NSGA.

## Abstract

Recently, the Internet of Things (IoT) has transformed the agriculture field, enabling farmers to cope with the major challenges in their daily life. A New inventions in the application of the Internet of Things try to solve problems by improving production parameters such as the quality, quantity, and sustainability of agriculture. Farmers are influencing this domain to monitor crop growth, soil moisture, etc. In addition, sensors can be used to remotely control irrigation equipment. Also, they easily make decisions about farm water supply, while taking into account functional parameters to reduce water consumption and response speed to user requests. In fact, to ensure the irrigation system customers need to construct a service composition. Further, to respond to both functional and non-functional parameters represented in quality of service (Qos) of the user needs. Currently, service composition (SC) is one of the major problems in the Internet of things due to the availability of atomic services that can serve similar functionality suitable. There is a set of candidates for each atomic service in SC from which a suitable candidate service is picked based on certain criteria. Quality of service (QoS) is one of the criteria to select the appropriate services. The present project transforms the service composition problem into an optimization problem. Moreover, in this project we also propose a combinatorial auction approach (the Or bids method) in order to provide the best services selection to construct the service composition.

**Keywords :** *Irrigation system , Service Composition, Atomic Services, QoS,Internet of things, Combinatorial auctions , Optimization method, NSGA-II.*

## Résumé

Récemment, l'Internet des objets (IoT) a transformé le domaine de l'agriculture, permettant aux agriculteurs de faire face aux principaux défis de leur vie quotidienne. A De nouvelles inventions dans l'application de l'Internet des objets tentent de résoudre les problèmes en améliorant les paramètres de production tels que la qualité, la quantité et la durabilité de l'agriculture. Les agriculteurs influencent ce domaine pour surveiller la croissance des cultures, l'humidité du sol, etc. En outre, les capteurs peuvent être utilisés pour contrôler à distance les équipements d'irrigation. Aussi, ils prennent facilement des décisions concernant l'approvisionnement en eau de l'exploitation, tout en tenant compte des paramètres fonctionnels pour réduire la consommation d'eau et la vitesse de réponse aux demandes des utilisateurs. En fait, pour assurer le système d'irrigation, les clients doivent construire une composition de services et pour répondre à la fois aux paramètres fonctionnels et non fonctionnels représentés dans la qualité de service (QoS) des besoins de l'utilisateur. Actuellement, la composition de services (SC) est l'un des principaux problèmes dans l'Internet des objets en raison de la disponibilité des services atomiques qui peuvent servir une fonctionnalité similaire appropriée. Il existe un ensemble de candidats pour chaque service atomique dans la composition de services, parmi lesquels un service candidat approprié est choisi en fonction de certains critères. La qualité de service (QoS) est l'un des critères de sélection des services appropriés. Le présent projet transforme le problème de composition des services en un problème d'optimisation. De plus, dans ce projet, nous proposons également une approche d'enchères combinatoires (la méthode Or bids) afin de fournir la meilleure sélection des services pour construire la composition de services.

**Mots clés :** *Système d'irrigation, Composition de services, Services atomiques, QoS, Internet des objets, Enchères combinatoires, Méthode d'optimisation, NSGA-II.*



# List of Figures

2.1	The Internet of Things . . . . .	5
2.2	The three layers architecture Burhan et al. [2018]. . . . .	6
2.3	The five layers architecture Burhan et al. [2018]. . . . .	8
2.4	Aggregate Network diagram with sensor network and access gateways Gigli et al. [2011] . . . . .	9
2.5	The atomic services for fire alarm . . . . .	11
3.1	Combinatorial Auctions . . . . .	16
3.2	Structure of the auctioneer . . . . .	18
3.3	single-sided Combinatorial Auction Zhang and Zhou [2020]. . . . .	20
3.4	Classification of meta-heuristic algorithms Chehouri [2018] . . . . .	21
3.5	Trajectory-based method . . . . .	22
3.6	Schematic of the NSGA-II procedure Jiang et al. [2021]. . . . .	23
4.1	The proposed architecture . . . . .	28
4.2	Sensing layer (collecting data from the field) . . . . .	29
4.3	Database Layer . . . . .	30
4.4	Treatment Layer . . . . .	31
4.5	Service Composition process . . . . .	32
4.6	Flowchart of NSGA-II algorithm . . . . .	34
4.7	The proposed genome - individual- . . . . .	35
4.8	An example of crossover operation . . . . .	38
5.1	Python . . . . .	42



5.2 Pycharm . . . . .	43
5.3 Microsoft Visio . . . . .	43
5.4 PyQt designer . . . . .	43
5.5 Service composition construction home page . . . . .	44
5.6 Atomic services components . . . . .	45
5.7 Service composition after Combinatorial auction selection . . . . .	46
5.8 Service composition after Combinatorial auction selection . . . . .	46
5.9 Best service composition . . . . .	47
5.10 Best service composition . . . . .	47
5.11 Pareto front of the first generation . . . . .	48
5.12 Pareto front of the last generation . . . . .	49
5.13 The final results of the status of the indicators . . . . .	49
5.14 The status of the indicators . . . . .	50
A.1 Check function . . . . .	58
A.2 Input function . . . . .	59
A.3 Start of the treatment . . . . .	59
A.4 Normalizing function . . . . .	60
A.5 Services function . . . . .	61
A.6 Combinatorial Auctions function . . . . .	62
A.7 Combinatorial Auctions function . . . . .	62
A.8 Service composition construction . . . . .	63
A.9 The crossover function . . . . .	64
A.10 Population Pt . . . . .	64
A.11 Fitness function . . . . .	65
A.12 Non dominance sorting function . . . . .	66
A.13 Fitness function . . . . .	66
A.14 Fitness function . . . . .	67
A.15 Crowding distance function . . . . .	67
A.16 Crowding distance function . . . . .	68

A.17 Pareto front function . . . . .	69
A.18 The main . . . . .	70
A.19 Mutation function . . . . .	70
A.20 Mutation results . . . . .	71
A.21 Crossover function . . . . .	71
A.22 The offspring population function . . . . .	72
A.23 The main . . . . .	72
A.24 The main . . . . .	73
A.25 The main . . . . .	73
A.26 The main . . . . .	74
A.27 The main . . . . .	74
A.28 The main . . . . .	75
A.29 The Status of the indicators . . . . .	76
A.30 An example of the last generation . . . . .	76

# List of source code

# List of Tables

- 3.1 Related work comparison . . . . . 25
- 5.1 The used sensors . . . . . 41
- 5.2 The used sensors . . . . . 48

# Contents

Acknowledgements . . . . .	i
Abstract . . . . .	iii
Résumé . . . . .	iv
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>1 General Introduction</b>	<b>1</b>
1.1 General context . . . . .	1
1.2 Problematic and Objectives . . . . .	2
1.3 Outlines . . . . .	3
<b>2 Service composition in IoT</b>	<b>4</b>
2.1 Introduction . . . . .	4
2.2 Internet of Things: IoT . . . . .	4
2.2.1 Definition . . . . .	5
2.2.2 Architecture . . . . .	6
2.2.3 IoT services . . . . .	8
2.2.4 Quality of services (Qos) . . . . .	10
2.3 Service composition : State of the art . . . . .	11
2.3.1 Atomic service . . . . .	11
2.3.2 Composite service . . . . .	12
2.3.3 Service composition process . . . . .	12
2.4 Conclusion . . . . .	13

<b>3</b>	<b>Combinatorial auctions</b>	<b>14</b>
3.1	Introduction . . . . .	14
3.2	Problem statement: IoT service composition . . . . .	14
3.3	Combinatorial auctions . . . . .	15
3.3.1	Definition . . . . .	15
3.3.2	Components . . . . .	16
3.3.3	Methods . . . . .	18
3.4	Used Algorithm . . . . .	20
3.4.1	Meta-heuristic algorithms . . . . .	20
3.4.2	Non-Dominated Sorting Genetic Algorithm II (NSGA-II) . . . . .	23
3.5	Related work . . . . .	24
3.6	Synthesis . . . . .	25
3.7	Conclusion . . . . .	26
<b>4</b>	<b>Design and Contribution</b>	<b>27</b>
4.1	Introduction . . . . .	27
4.2	Proposed architecture . . . . .	27
4.2.1	Architecture description . . . . .	28
4.2.2	Service composition process . . . . .	31
4.2.3	Used algorithm . . . . .	38
4.3	Conclusion . . . . .	39
<b>5</b>	<b>Implementation and results</b>	<b>40</b>
5.1	Introduction . . . . .	40
5.2	Development tools and used platforms . . . . .	40
5.2.1	Hardware: . . . . .	41
5.2.2	Software: . . . . .	42
5.2.3	PyQt designer . . . . .	43
5.3	System interfaces . . . . .	43
5.3.1	Home page . . . . .	44
5.3.2	Atomic services components . . . . .	44

5.3.3 Combinatorial Auctions selection . . . . .	45
5.3.4 Best service composition . . . . .	46
5.4 Obtained results and discussion . . . . .	47
5.5 Conclusion . . . . .	50
<b>6 Conclusion and Perspectives</b>	<b>51</b>
6.1 Conclusion . . . . .	51
6.2 Perspectives . . . . .	52
<b>References</b>	<b>53</b>
<b>A Appendix</b>	<b>58</b>

# Chapter 1

## General Introduction

### 1.1 General context

In today's world, with the rapid growth of the world's population, agriculture is becoming more and more important to meet human needs. However, agriculture requires irrigation but with every year we have more water consumption than rainfall. For farmers, finding ways to save water while obtaining the highest yields has become crucial. But in the present era, farmers have been using irrigation techniques to irrigate the land regularly. By using Internet of Things (IoT) and sensor network technology we can control water wastage and to maximize the scientific technologies in irrigation methods. Hence, it can greatly increase water consumption and increase water productivity.

Recently, IoT has caused a stir with the rapid development of microchips, sensor devices, networks and software. According to Forbes, *"it's a concept that not only has the potential to affect our lives, but also the way we work"* Morgan [2017]. In fact, this combination will make field monitoring much easier than ever. In our work we propose a problematic in the the smart irrigation system, because this sector has known a real digital revolution for several years to take economic and social aspects into account and current environmental problems.

Thus ,the main goal of IoT is therefore to connect a large number of everyday objects to the Internet, to give them their own identity, to enable them to provide functions and to collect information in the form of services. Therefore, most IoT devices cannot function on their own and must perform a variety of service combinations and define them as a service composition. It includes the function of combining multiple services in a single process to meet



complex requirements that a single service cannot. In fact, This composition takes functional and non-functional parameters into account, in particular with regard to the quality of service (QoS) required by the user.

In addition, the composition of the services poses a significant problem namely the selection of composite services that are sensitive to QoS (Quality of Service) with support for user parameters. As a consequence, a new approach named combinatorial auction has emerged. This approach is responsible for the selection of atomic services based on the quality of the service. This last is viewed as a metric that can help customers to determine the best composition of IoT services for their applications and optimize the quality of the service .

## 1.2 Problematic and Objectives

The agriculture has been recently transformed by Internet of Things (IoT) that enables farmers with enormous challenges they face in daily life. Farmers influence IoT to monitor the crop growth, soil moisture etc...by using a huge number of sensors,to manage and control irrigation equipment remotely and to make decisions easily. And so on,they can also effectively open and close water gates to cover a large area without wastage of water through the apps on their smart phones [Anandkumar et al. \[2018\]](#).IoT connected sensors inform the farmer of yield status.

Due to the variety and the exponential growth of the services used depending on the needs of the users, the composition of the services is currently a major problem in the Internet of Things. So classical composition approaches tried to offer customers composite services. However, the non-functional aspect of these (composite) services is not taken into account. This aspect is mainly related to the values of the quality of the services. All these problematics are dealt with separately in the literature. Hence, in our work we had a challenge in terms of solving the problems of the irrigation system, which is a step towards agricultural growth taking into account the Qos (quality of service) of the sensors present within the farm (including energy, reputation, response) in time ),it is main objectives are :

- To reach and cover a large area with the water .

- To extract the best services among the important number of services .
- To ensure the best configuration with best sensors in term of quality .

## **1.3 Outlines**

Our work is organized as follows:

In the second Chapter we talk about state of the art of the Internet of Things (IoT). We will start with some definitions of the Internet of Things and its architecture, as well as some Iot services. We also show the behavior models of atomic services. Based on this, we specify the composition of the service in the Internet of Things and discuss some important concepts such as the service composition process.

In the third chapter we discuss some related work in in Iot service composition. People have been working in this field since the dawn of IoT to solve various problems, then we define the approach used in our project which is the combinatorial auction, its components , and we highlight their various methods and indicate the one used. At the end, we present some meta heuristic algorithms .

The fourth chapter is mainly about design and contribution, we present our architecture proposal to solve the service composition problem in a smart irrigation system, the proposed architecture is explained along with the UML sequence diagram to make our work easier to understand.

The implementation and the results are discussed in the five chapter, we introduce the development tools and frameworks, with screenshots of the system , then we discuss and analyse the obtained results.

Finally, we conclude our work in the sixth chapter.

# Chapter 2

## Service composition in IoT

### 2.1 Introduction

Nowadays, the Internet is everywhere, covering almost every corner of the world. It affects human life in incredible ways. However, the journey is far from over. We are now entering an era where we pay more attention to various household appliances, consequently, we are going into the internet of things era.

The Internet of Things (IoT) is an emerging technology that can transform industry, the environment, society and medicine. Moreover the Internet of Things is the integration of information space and physical space, which consist of the interconnection of on-board computers, sensors, mobile devices or other uniquely identifiable objects. In fact, this interaction is shown in collaboration with its environment in order to achieve common goals through the use of an existing Internet infrastructure [Sethi and Sarangi \[2017\]](#). The purpose of this chapter is to introduce the Internet of Things, the composition of services and their existing methods, and to describe and compare some of the key requirements. The last section concludes this chapter.

### 2.2 Internet of Things: IoT

Human life has experienced different waves which are agriculture, industry and information technology [Jamali et al. \[2020\]](#). However, there seems to be a new wave in our lives. Nowadays, new technology makes it possible for anyone to connect anything anytime, anywhere. This

technology is called the "Internet of Things". In this section, we will elaborate in detail what is the Internet of Things, its architecture and its services.

### 2.2.1 Definition

According to Rayes and Salam the Internet of Things can be defined as follows: "*the network of Things with clearly identifiable elements that integrates software intelligence, sensors, and ubiquitous Internet connections*" [Rayes and Salam \[2017\]](#).

For ease of definition, the term IoT can be regarded as an IoT system in the literature, which refers to a set of devices and middleware used to manage the interaction between devices and networks. Engineering includes issues related to developing technologies to build IoT systems. Further, The proposed software method supports the development of IoT systems. In addition, IoT has potential to build up various applications in diverse areas of automotive industry, medical and healthcare, telecommunications, agriculture, retail, and supply chain management. [Jamali et al. \[2020\]](#).

At present, the Internet of Things is considered to be an integral part of almost all intelligent systems due to its huge impact on various fields. This claim is powered by McKinsey's report on the global economic impact of the Internet of Things. The annual economic impact of the Internet of Things in 2025 will be between 2.7 and 6 \$ [Manyika et al. \[2013\]](#).



Figure 2.1: The Internet of Things

## 2.2.2 Architecture

There is no consensus on the generally accepted IoT architecture. Different researchers have proposed different architectures. Three-tier and five-tier architecture. The most basic architecture is a three-tier architecture, as shown in 2.2, it was introduced in the early stages of research in this area. It consists of three layers, namely the perception layer, the network layer and the application layer [Sethi and Sarangi \[2017\]](#), [Patel et al. \[2016\]](#).

- **The perception layer :** Is the physical layer, which has sensors for sensing and gathering information about the environment [Patel et al. \[2016\]](#). It senses some physical parameters or identifies other smart objects in the environment [Sethi and Sarangi \[2017\]](#).
- **The network layer :** Is responsible for connecting to other smart things, network devices, and servers. Its features are also used for transmitting and processing sensor data [Sethi and Sarangi \[2017\]](#).
- **The application layer :** Is responsible for providing users with application-specific services, defining various applications that can be used by the Internet of Things, for example smart homes, smart cities, and smart health [Sethi and Sarangi \[2017\]](#).

IEEE standard association considers three layers for IoT architecture [Burhan et al. \[2018\]](#), as explained below:

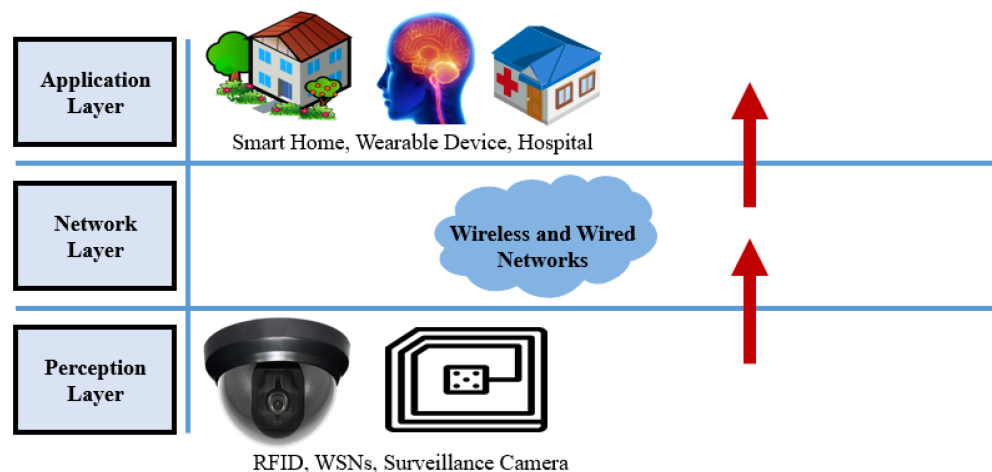


Figure 2.2: The three layers architecture [Burhan et al. \[2018\]](#).

The three layers architecture defines the basic idea of the Internet of Things, but it is not enough for research, because it often focuses on more finer aspects. In order to improve the previous architecture, they invented more layers, one of which is five-layers architecture that also includes the processing layer and the business layer.

The five layer are perception, transport, processing, application and business these levels are illustrated in figure 2.3 according to [Antão et al. \[2018\]](#). The functions of the perception layer and the application layer are the same as the three-tier architecture. We outline the function of the remaining three layers [Sethi and Sarangi \[2017\]](#), [Zouai et al. \[2019\]](#).

### **The five layers architecture**

- **The transport layer :** Transfers the sensor data from the perception layer to the processing layer and vice versa through networks such as wireless, 3G, LAN, Bluetooth, RFID, and NFC [Sethi and Sarangi \[2017\]](#).
- **The processing layer :** Is also known as the middleware layer. It stores, analyzes, and processes huge amounts of data that comes from the transport layer. It can manage and provide a diverse set of services to the lower layers. It employs many technologies such as databases, Cloud Computing, and Big Data processing modules [Sethi and Sarangi \[2017\]](#).
- **The business layer :** Manages the whole IoT system, including applications, business and profit models, and user's privacy [Sethi and Sarangi \[2017\]](#).

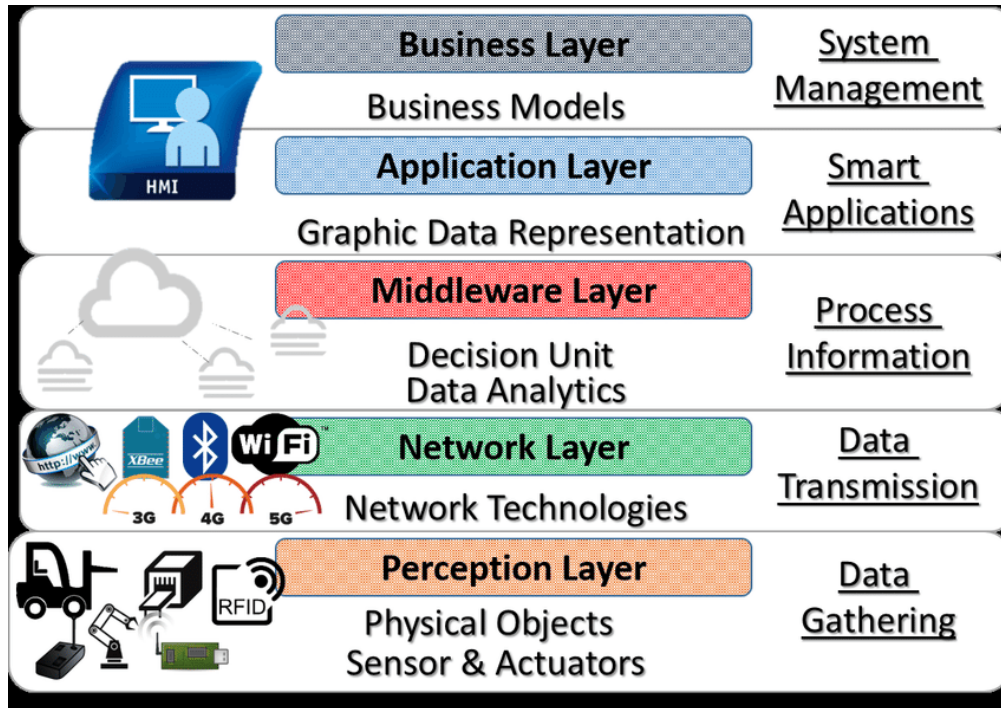


Figure 2.3: The five layers architecture [Burhan et al. \[2018\]](#).

### 2.2.3 IoT services

Specific IoT services can be used to simplify application development and accelerate deployment. Note that the following categories apply to the following services:

1. **Information Aggregation Services** : They involve the process of receiving data from various sensors, processing data, and transmitting data to applications through the Internet of Things. figure 2.4 illustrate the aggregate Network diagram with sensor network and access gateways.

Information aggregation services do not need to implement a certain communication channel for collaboration [Gigli et al. \[2011\]](#). When using an access gateway, information aggregation services can use different types of sensors and network devices, and exchange data through public services. For example, an application can use RFID tags to identify the identity of certain devices, and use the ZigBee network to collect sensor data, and then use a gateway device to relay this information to the application on the same service [Xiaojiang et al. \[2020\]](#).

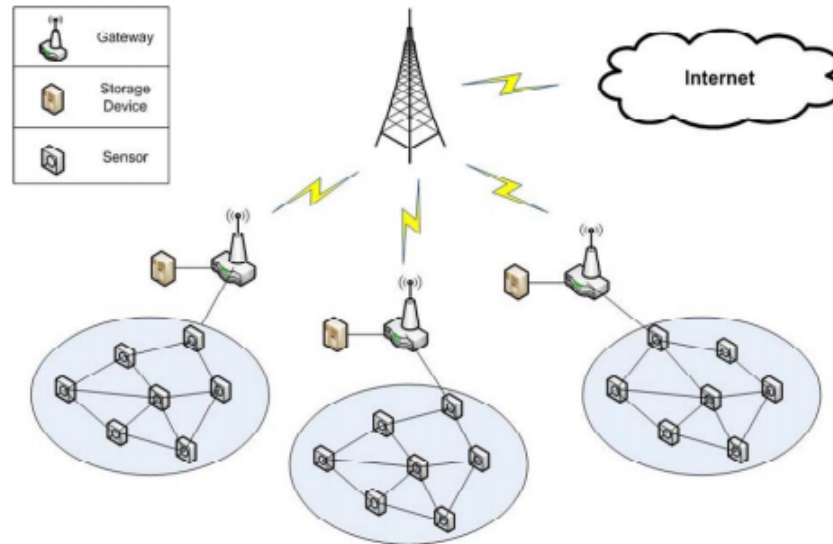


Figure 2.4: Aggregate Network diagram with sensor network and access gateways [Gigli et al. \[2011\]](#)

2. **Collaborative aware services :** These services use aggregated data to make decisions and take actions based on those decisions. With the formation of the Internet of Things, it should lead to the development of complex services that use any data that can be retrieved from a huge network. This requires not only the ability to extract information, but also the transmission of responses to the collected information in order to take action. Therefore, these services require end-to-end communication and end-to-person communication [Garcia-de Prado et al. \[2017\]](#). The Internet of Things infrastructure certainly requires higher reliability and speed, and requires the terminal to have more computing power or connect to another device that performs this operation
3. **Medical treatment :** Devices developed by the Internet of Things can help patients with treatment. On the one hand, the hospital's treatment costs are too high, on the other hand, through the use of these IoT devices, it can be affordable to patients to continue treatment at a lower cost. The most common equipment in this field is used to combat diabetes [Iot \[2021\]](#).
4. **Detecting Machine failure :** Nowadays, the use of machines is too difficult to understand. With the help of the Internet of Things, a system that can detect machine failures can be



developed. These machines are used to warn users that certain parts of the system are not working properly that will be helpful to ensure product quality and can also prevent fatal accidents for machine users [Iot \[2021\]](#).

5. **Integration with AI application** : The next important step is artificial intelligence, because almost all smart devices use it to improve efficiency. The concepts and functions of the Internet of Things can be integrated into artificial intelligence applications for better performance and processing power. The market uses both artificial intelligence and IoT applications, and these devices are already operating efficiently [Iot \[2021\]](#).

#### 2.2.4 Quality of services (Qos)

Since we have many QoS values used to evaluate IoT services, we show the used ones in this section. These QoS are a sort of indicators, also called non-functional parameters. The QoS values used in our project are defined as follows:

- **Response time** : The most important goal in service delivery is to deliver services to consumers in a timely manner. Response time is measured by several sub-factors, such as the average response time and the maximum response time promised by the service provider [Merizig et al. \[2019\]](#).
- **Reputation** : Reputation is the level of confidence that users have to accept the service. It measures the trust that the service has gained based on previous experience [Merizig et al. \[2019\]](#).
- **Distance** : The distance is a numerical measurement of how far apart the sensors from the area we want to supply it with the water.
- **Energy** : This is the level of charge generation that is proportional to the pulse energy. Generally, active sensors require external energy for their operation, named control signal, which is used by the sensor to produce the output signal.

## 2.3 Service composition : State of the art

In this section, we first show the behavioral model of atomic services, and on this basis, we define the composition of services in the Internet of Things.

### 2.3.1 Atomic service

The behavior of atomic services is determined by the sequence of operations in which operations interact with each other (receiving and sending messages). Multiple service operations with different names can provide the same function [Bhalerao and Ingle \[2019\]](#).

As an example of atomic services we illustrate the fire alarm service in the conference room. In this example, we assume that there are multiple devices in the conference room, including temperature sensors, smoke detectors, alarm bells, and sprinklers. They are considered as atomic services and are described in the figure 2.5. The state is represented by a circle, and the transition is represented by an arrow. Each process is identified by its own process type [Li et al. \[2012\]](#).

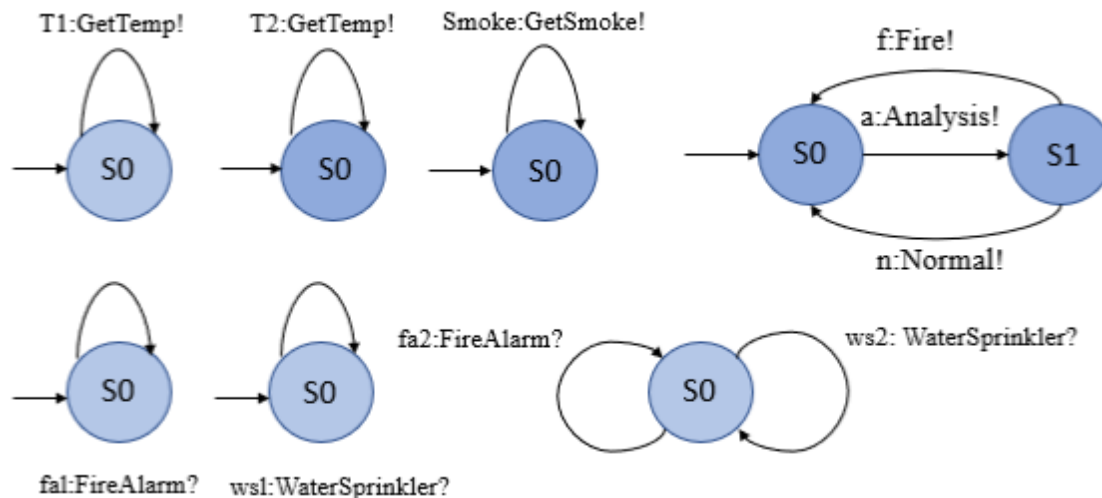


Figure 2.5: The atomic services for fire alarm

### 2.3.2 Composite service

A composite service is an assembly of atomic or other composite services. The ability to assemble services is referred to as composability. Composite services are also referred to as compound services [Arellanes and Lau \[2020\]](#). Generally, a composite service would be seen as coarse grained or having a larger granularity.

At runtime, each operation of the business process is dynamically delegated to available atomic services. The service composition in IoT can be specified to represent the combination of the behavior model of business process and atomic service [Li et al. \[2012\]](#).

### 2.3.3 Service composition process

Implementing an application through service composition involves several steps, enabling you to gradually move from an abstract specification to a specific composition of services, meaning the combination that is ready to be executed.

1. **Description phase :** At this stage, the user enters a request containing the required service information [Merizig \[2018\]](#), and then considering the quality of the service provided by the user (the request contains different data), these requests will be automatically decomposed by the ontology or other models.
2. **Discovery and Selection phase :** At this stage, the necessary services are defined to meet the previously determined functional requirements. Searching in the service registry will find services that match the capabilities of each abstract service in the composition. This search is usually based on the description (syntax or semantics) of the available services [Shehu et al. \[2014\]](#). The result of the discovery phase usually consists of several candidate services with similar functions but different non-functional attributes (QoS). The selection stage can determine the most suitable combination of services among all previously determined candidate services based on non-functional characteristics .

In this step, in each action of the composite service, we will search for the service type corresponding to each action in the service registry according to the description of the provided service [Merizig \[2018\]](#). Thousands of services can be found at this stage. A matching

algorithm must be used to select the correct service for a specific operation. At the end of this step, we can combine a series of services to meet the needs of users.

3. **Deployment phase :** At this point, the composite service is implemented for end users to use in the invocation and instantiation operations. At the end of this step, we have an executable composite service [Merizig \[2018\]](#).
4. **Execution phase :** At this point, a composite service instance is created and executed by the runtime, which is responsible for calling various components of the service. Perform various operations during execution [Merizig \[2018\]](#). Automatic dialing methods can be found that merge the first two steps into one step. In this way, the created composite service is created directly based on the linkage conditions, and there is no need to create an abstract composite service.

## **2.4 Conclusion**

The Internet of Things (IoT) describes the entire system of networked physical things. In addition, the service composition process makes it possible to interact between user needs and smart objects in the IoT environment.

Leveraging on the service discovery process can influence the search for the desired services. Therefore, choosing the right service is the biggest challenge, including the required features and quality. Finally, this chapter outlines the Internet of Things and its services. Based on this, we demonstrated the service design methods and processes used in the IoT environment. In the next chapter, we will focus on combinatorial auctions.

# Chapter 3

## Combinatorial auctions

### 3.1 Introduction

Auction is a vital market mechanism that has been used to distribute goods and services since the earliest times. Additionally they are favored by public and private institutions, especially in the general transaction process (lottery tickets, price announcements..etc), as a result of they are open and fairly fair, it is usually easy to understand by the participants and usually brings economically effective results [Vangerven \[2017\]](#).

However, their true popularity has only been observed in the most recent period (about ten years ago) [Abrache et al. \[2007\]](#). Due to the emergence of e-commerce and changes in the growth trend of important business processes on the Internet, as well as the wave of privatization and deregulation leading to various industries.

Therefore, this chapter focuses on combinatorial auctions and their components, as well as methods and evolutionary algorithms for finding solutions to the problems raised in our work.

### 3.2 Problem statement: IoT service composition

Services are the foundation of the IoT paradigm because they allow you to ascertain the capabilities of connected objects and interact with them. In addition, these services enable Internet users to define their requirements for these objects and their functions. Furthermore, the requirements may become more complex and require multiple services to complete an request. As a result, this has led to the definition of requirements for complex interactions between cer-

tain objects.

A service is said to be composite if its execution requires several other services to invoke its functionalities. Otherwise it is called an atomic service. User needs can be defined as a service composition problem. In order to improve the smart irrigation system and protect the interests of farmers (minimize energy consumption and water waste during the irrigation process), our project aims to transform this problem into a series of services ( service composition ), taking into account the the nonfunctional parameters (Qos values).

### 3.3 Combinatorial auctions

Combinatorial auctions are an important interdisciplinary field that brings together the topics of economics, game theory, optimization, and computer science. They were first planned for the allocation of airport landing slots by *Russenti Smith*, and *Bulfin* (1982). As a consequence, this section reviews the definition of Combinatorial auctions with a well-balanced view integrating their components and methods.

#### 3.3.1 Definition

According to [Voudouris \[2020\]](#) "*Combinatorial auctions are a smart market where bidders can bid on a combination of individual different items or "packages", rather than a single item or continuous quantity. These packages can also be called batches, and the auction ends with multiple batch auctions.*"

Combinatorial auctions apply to bidders' non-additive ratings for sets of items (that is, their ratings for the combinations of items are higher or lower than the sum of their ratings for the individual items in the combinations). Although they allow bidders to be more expressive. Compared with traditional auctions, combinatorial auctions are both a computational problem and a game theory problem [Beall et al. \[2003\]](#),[Zhong et al. \[2020\]](#).

The simple combinatorial auction shown in [Figure 3.2](#) has been used in real estate auctions for several years. wherever a typical procedure is to simply accept bids for bundled goods [Zhong et al. \[2020\]](#). Recently, they have been used in trucks, bus routes, factories, and radio spectrum allocation for wireless communications.

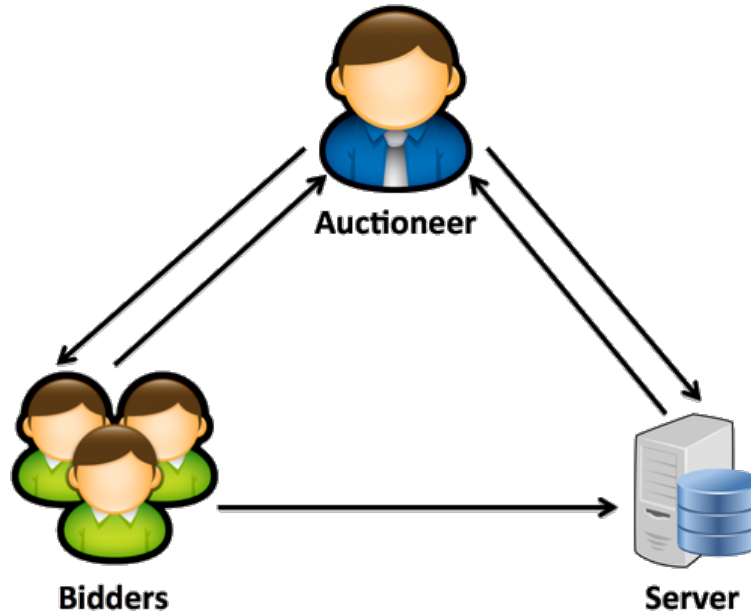


Figure 3.1: Combinatorial Auctions

### 3.3.2 Components

The overall system consists of three components: *the buyers*, *a seller*, and *an auctioneer*. Buyers, sellers, and auctioneers are the three main participants in the auction process. Users act as sellers when they sell their load shedding capabilities, and the grid becomes a buyer, with a certain incentive to purchase users' load shedding capabilities. Although, the auctioneer is the central person who trade this exchange between the user and the grid.

At the beginning of the auction, the auctioneer collects bids from the grid and the user and arranges them according to their order layout. After bids initialization, winners are selected using the WDP. The WDP of the combinatorial auctions improves the overall social welfare [Zaidi et al. \[2021\]](#).

#### 1. Structure of the Auctioneer :

The auctioneer is the organization responsible for collecting bids from buyers and sellers and comparing them accordingly. It is also responsible for calculating the price of the transaction [Zaidi et al. \[2018\]](#) as shown in Figure 3.2. The key components of auctioneer according to [Zaidi et al. \[2021\]](#) are as follows:

- (a) **Market Communication Manager :** Used for communication between auctioneers and bidders. It collects bids, informs bidders about the results, and communicates with the matching module through the order book and exit manager.
- (b) **New Bid Clock :** Pays attention to the new price. When the time is up, the winner will be announced and the auction round will end. Return to the initial stage, whenever there is a new bid is received
- (c) **New Winner Clock :** Focus on the new winners. It will be updated every time a new winner appears (buyer and sellers selected for trade).
- (d) **Matching Module :** Runs the algorithm to determine the winner and select the winner. Looks for a new winner before the end of the auction round. Retrieve bids from the market data set and report the results through the Outbound and Communication Manager.
- (e) **Order Book :** All the bids are collected in the order book and remains there until they win or are expired.
- (f) **Market Output Manager :** Retrieve the result from the matching module and store it in the market record, and provide the result to the user through the market link manager.
- (g) **Market Log:** Keeps the history of the market trades, all the winning and non-winning bids which (valid and expired bids) via order book and output manager. Provides historical data to the users and the grid



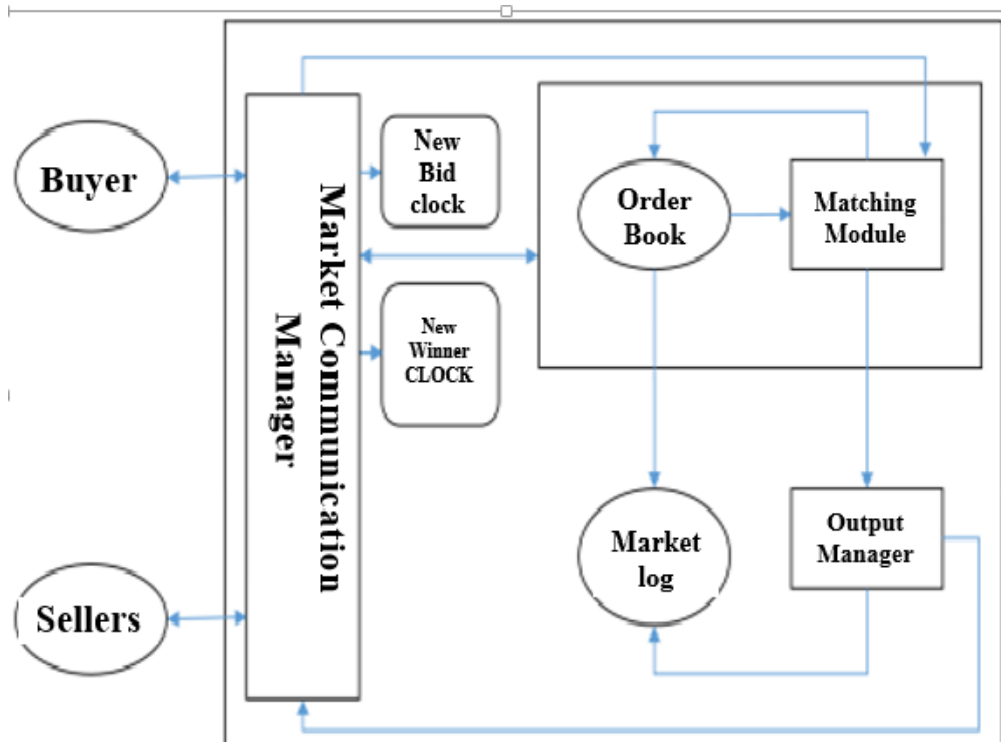


Figure 3.2: Structure of the auctioneer

### 3.3.3 Methods

There are different methods of auctions that can be categorised by different criteria [Ramírez et al. \[2014\]](#) presented as follows :

- The number of participants.
- The divisible or discrete nature of the item.
- The number of items sold.
- The items are identical or distinct.
- Whether the bids are known by the rest of participants or not.
- Whether the price finally paid is the highest price of the bids or the second highest price.

According to that we are going to mention the two main methods:

## 1- The combinatorial double auction

A double auction is the process of buying and selling items involving multiple sellers and multiple buyers. Potential buyers bid, and potential sellers submit their selling prices to the market organization, and then the market organization chooses the price  $p$  at that clears the market. A typical example of a double auction is the stock exchange [Kiedanski et al. \[2020\]](#). Traditionally there was a market for every good, and the price path that determines equilibrium prices has been described as a groping process (Walras,1874) [Korzhenevych \[2017\]](#).

This process is based on the assumption of approximate substitutability between commodities, which obviously does not exist in the commodity market. Therefore, a groping process will not lead to the establishment of equilibrium prices for these commodities.

In addition, these processes also assume that the number of goods sold is divisible [Xia et al. \[2005\]](#). Although in practice this is not necessary. In other words, many commodities have to be traded in integral units, fractional quantities yield useless results. Combinatorial double auctions solve both of these problems [Tafsiri and Yousefi \[2018\]](#).

## 2- The combinatorial single auction

In our case we are going to work with a special case of (CDA) which is :

- **Single sided auctions:** Or auctions with individual sellers/buyers are a some special cases of (CDA) .There is only one package on the market that can provide/buy all products [Devi et al. \[2021\]](#). Another definition was giving by [Jung et al. \[2013\]](#) that one seller is selling several goods to multiple buyers as shown in Figure 3.3.

A special case of the single-sided auction is Auctions with unique items means there is only one unit of each component up for auction. Or, in the case of a unique seller [Devi et al. \[2021\]](#).

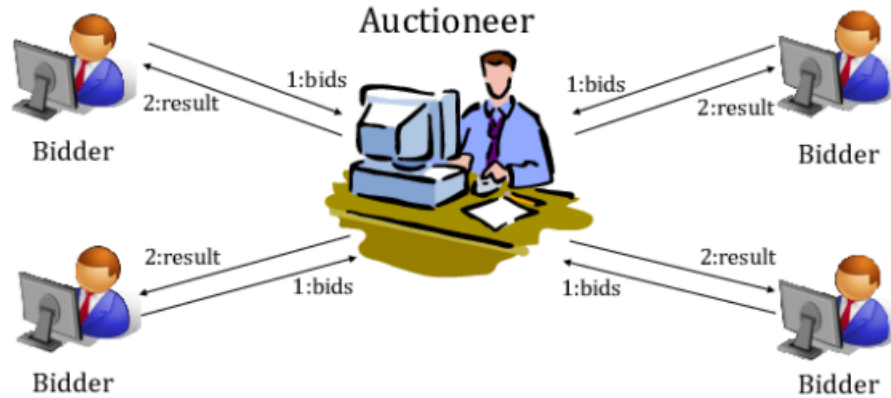


Figure 3.3: single-sided Combinatorial Auction [Zhang and Zhou \[2020\]](#).

## 3.4 Used Algorithm

In this section we introduce some meta heuristic algorithms that used to optimize our solution.

### 3.4.1 Meta-heuristic algorithms

According to [Lazar and Reynolds \[2003\]](#) *The words of “Meta” and “Heuristic” are Greek where, “meta” is “higher level” or “beyond” and heuristics means “to find”, “to know”, “to guide an investigation” or “to discover”*. Heuristics [Jahwar and Abdulazeez \[2020\]](#) are methods to find a good (almost) optimal solution through reasonable calculation costs, but do not guarantee feasibility or optimization. In other words, meta-heuristics are a set of intelligent strategies used to improve the efficiency of the heuristic process.

According to [Wu \[2021\]](#) the Meta-heuristics is an iterative core process that monitors and modifies the performance of subordinate heuristics in order to efficiently generate high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions per iteration. Sub-heuristics can be high-level (or low-level) processes, simple local searches, or just construction methods.

#### 1. Classification :

Many methods to classify meta-heuristics based on selected features have been proposed,

as shown in Figure 3.4. In this section, we briefly summarize the population-based and single point search.

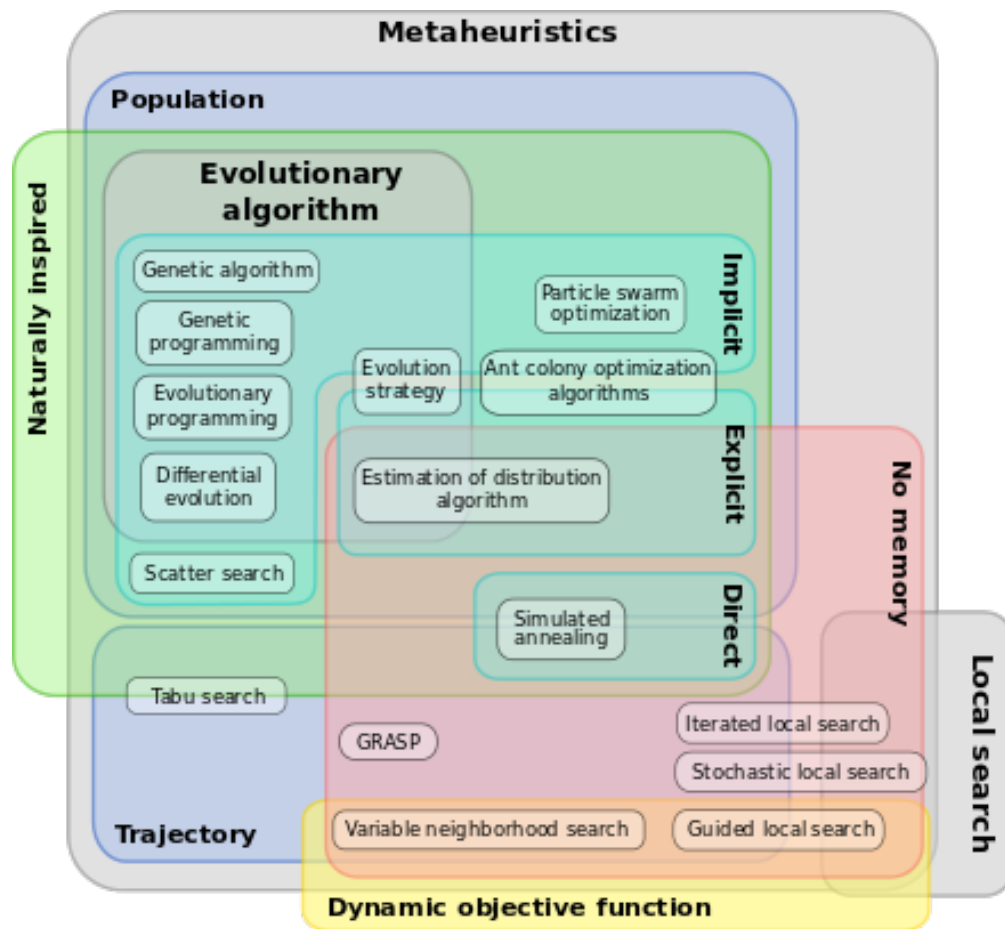


Figure 3.4: Classification of meta-heuristic algorithms [Chehouri \[2018\]](#)

It can also be classified according to the number of solutions used at the same time. Path methods, as shown in Figure 3.5 are algorithms that are always based on a single solution. [Ahmed et al. \[2018\]](#)

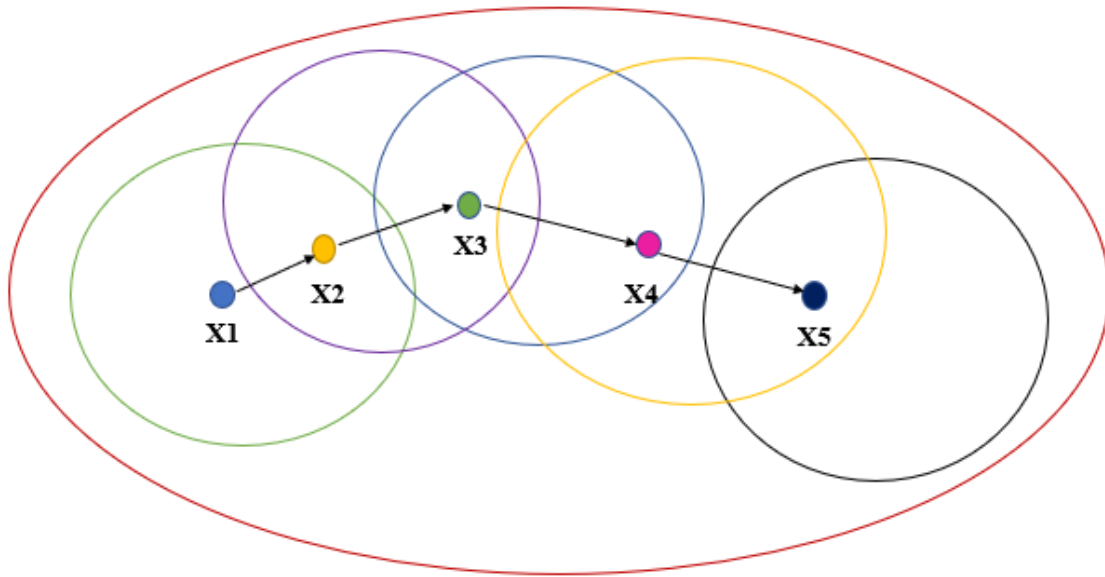


Figure 3.5: Trajectory-based method

- **Single-based meta-heuristic algorithm :**

For single-based algorithms, the search ends at the starting point. It will explore its neighbourhood with a set of moves to improve their solution [Samsuddin et al. \[2018\]](#). Single solution element heuristics include simulated annealing (SA), tabu search (TS), variable neighbourhood search (VNS) and greedy randomized adaptive search procedure (GRASP).

- **Population-based meta-heuristic algorithm :**

It can use multiple starting points in parallel to complete the search process. The advantage of the population-based algorithm is that it can effectively provide a search space for exploration. This method is suitable for global search because it includes the global and local search function [Samsuddin et al. \[2018\]](#). These include particle swarm optimization, genetic algorithm (GA), evolutionary strategy (ES), intelligent raindrop algorithm and artificial bee colony (ABC) and ant colony optimization (ACO).

Therefore, we will use a population based algorithm that will briefly explain in the next subsection which is: **the Pareto Dominance Relationship** exactly **the (NSGA-II)**.

### 3.4.2 Non-Dominated Sorting Genetic Algorithm II (NSGA-II)

NSGAI is a cross-generation genetic algorithm and one of the most popular general optimization algorithms. It is widely and successfully used in many practical applications [Yusuf et al. \[2021\]](#).

It is one of the first multi-standard algorithms to introduce elitism, which means that the elites in the crowd have a chance to be carried over to the next generation. It uses a fast non-dominated sorting procedure based on Pareto front ranking [Mahrach et al. \[2020\]](#).

As an attempt to convergence, meaning, emphasizing non-dominated solution. In addition to the reasons given above, we have selected this algorithm because it uses a clear mechanism to maintain diversity (Distance between crowds). According to [Jiang et al. \[2021\]](#) the next figure 3.6 shows the work of NSGA-II and its basic steps are as follows:

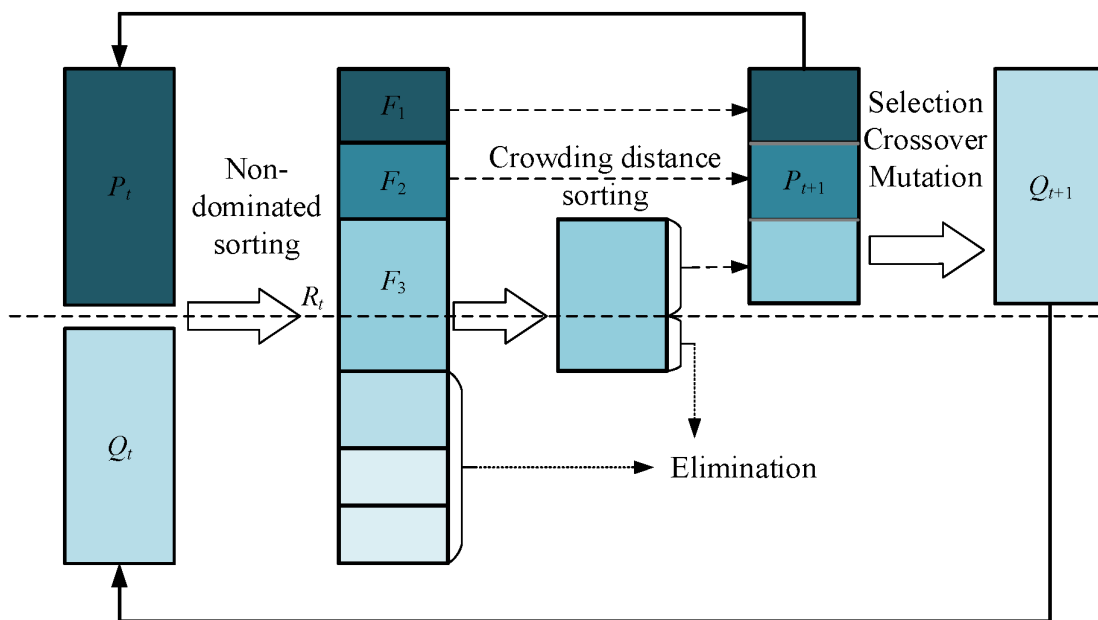


Figure 3.6: Schematic of the NSGA-II procedure [Jiang et al. \[2021\]](#).

The steps of NSGA-II depicted in the previous figure are described as follow:

1. According to the scope and constraints of the problem, an initial population of size  $N$  ( $t = 0$ ) is randomly generated. Moreover, we use the selection, crossover, and mutation operator GA (duplication of the initial population  $2N$ ) to generate the offspring population

- $Q_t$  ( $t = 0$ ) from  $P_t$ . Then calculate the fitness score of each person.
2. Combine the  $P_t$  and  $Q_t$  populations to create a new  $2N$  population. . Perform an undominated classification of the population  $R_t$  and classify it on multiple fronts (F1, F2, F3...). Then calculate the crowding distance for a set of individuals in the population.
  3. According to the order of dominance of the front and the crowding distance,  $N$  best individuals are selected to form a new population of  $P_{t+1}$ .
  4. Perform selection, crossing, and mutation in the  $P_{t+1}$  population to produce a new  $Q_{t+1}$  offspring of size  $N$ .

### 3.5 Related work

Many works in the literature has attempted to solve the problem of service composition in IoT through the implementation of various approaches,including the multi-agent system, linear programming, exact methods, and evolutionary algorithms.Our study to some works relevant aim at solving this problem.

[Baker et al. \[2017\]](#)present and evaluate a novel Energy-aware multi-Cloud IoT service composition algorithm, the authors develop a novel multi-cloud IoT service composition algorithm called (E2C2) to solve this problematic. The algorithm was evaluated against four established service composition algorithms in multiple cloud environments (All clouds, Base cloud, Smart cloud, and COM2). However,this work generates energy efficient composition plans by integrating the least possible number of services from service providers.

In the work presented by [Merizig et al. \[2019\]](#) propose a solution to solve the deployment problem by considering it as a service composition taking into account the quality of service,he proposes a multi-agent system architecture to solve the deployment problem through service composition.In addition,they have used the Evolutionary Niche Pareto Genetic Algorithm - NPGA- as a solution to this problematic and to reduce the computation during the execution.Also in this work the authors take into consideration the non-functional aspect.The authors focus exclusively on cloud computing as an application area of their work. However, we see that the dead end of this work arises in the selection of the Niche Pareto.

Yuan et al. [2020] proposed an approach to solve the problem of service composition in the cloud manufacturing. This method is based on requirements tasks and the combination of quality of service objectives. But they eliminated the targets reputation and availability.

### 3.6 Synthesis

In this section, we will compare the work mentioned in this section chapter, the comparison is based on four different parameters: The QoS values are mentioned or not mentioned and the limits. We also have expanded the areas of application and the methods used for every job which are the main point in our work due to its importance, especially in the decision-making process.

Table 3.1: Related work comparison

	<a href="#">Baker et al. [2017]</a>	<a href="#">Yuan et al. [2020]</a>	<a href="#">Merizig et al. [2019]</a>
<b>Used method</b>	E2C2 algorithm	Mathematical model and the Evaluation method	the Evolutionary Niche Pareto Genetic Algorithm -NPGA
<b>Values of QoS</b>	mentioned	mentioned	mentioned
<b>Purpose of the work</b>	solving the problem of energy the in cloud	the service composition in cloud Manufacturing	deployment of services in cloud
<b>Limits</b>	Lack of method of selecting the best atomic service candidates from service provider list.	Absence of some QoS objectives	Selection according to Niche Pareto

To heal the previous mentioned limits from related works, our solution consists of using a greedy method to select the best service candidates. To ensure the best solution for the user we take into consideration the QoS values in service composition process. To handle the selection



of the best service composition it is better to consider the whole possible solution from the service collection in the contrary with NPGA algorithm.

### **3.7 Conclusion**

Both the research and the practice of combinatorial auctions have grown rapidly over the past ten years. In this chapter, we gathered and discussed some interesting knowledge about combinatorial auctions and the main uses of this last one in our project. We also explained the method used of combinatorial auction and the algorithm used in order to make it accessible to solve the smart irrigation system. In the next chapter we will present the design and contribution of our work .

# Chapter 4

## Design and Contribution

### 4.1 Introduction

After presenting the current status of the project, this chapter introduces the modeling and design of suggested ideas for solving irrigation system problems through service composition.

The purpose of service composition is to determine a combination of services based on a request from a user (client). Therefore, our goal is to build a robust system that intelligently provides a smart irrigation system based on by combining artificial intelligence with IoT through sensors deployed in the field. In this chapter we will discuss the architecture of our proposal to solve the problem in hand. Next we explain the role of each layer in detail, including the process of our architecture.

### 4.2 Proposed architecture

In order to achieve our goal, we proposed an architecture made up of three layers that interact with each other and exchange data in real time. Each layer is responsible for a specific task that generates data for the next layer starting from the physical field (the farm). The first layer captures the data as numerical values of the cultivation field and then forwards them to the second layer, which is regarded as a storage that provides us with the data. The third layer is used for data processing, it Chooses the best mix of services from a range of services while doing real-time data locally. The entire system architecture is shown in the following figure [4.1](#)

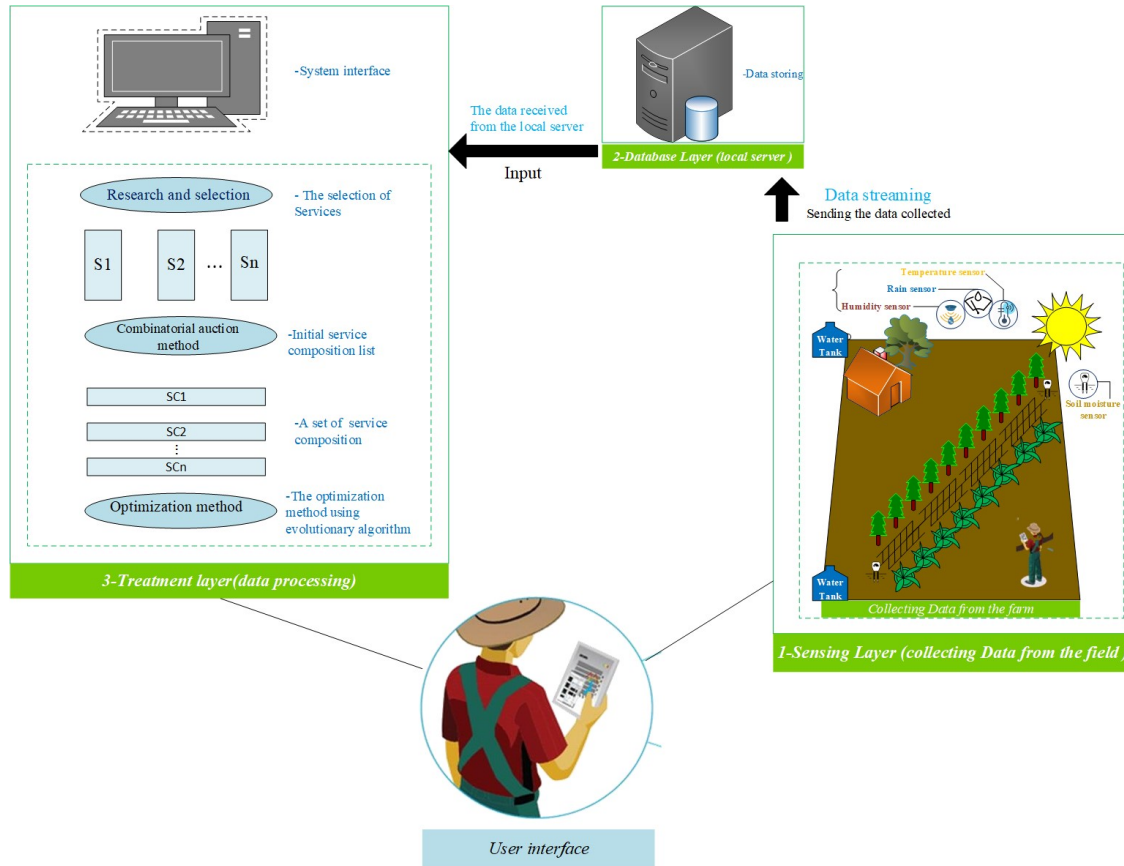


Figure 4.1: The proposed architecture

The previous architecture consist of several component which will described in next subsection.

#### 4.2.1 Architecture description

1. **Sensing layer** : This layer represents the physical part of the system. It is responsible for sensing the environment of the farm and reporting back to the server. As mentioned above, the sensing process is achieved by the large number of sensors deployed on site (temperature sensor, moisture sensor, soil moisture sensor and rain sensor) and a set of indicators which, ensure the irrigation system. Moreover, the farm is constantly monitored, resulting in the generation of large amounts of Qos values represented in the energy and reputation, response time and distance that will transmitted on to the next layer for storage.

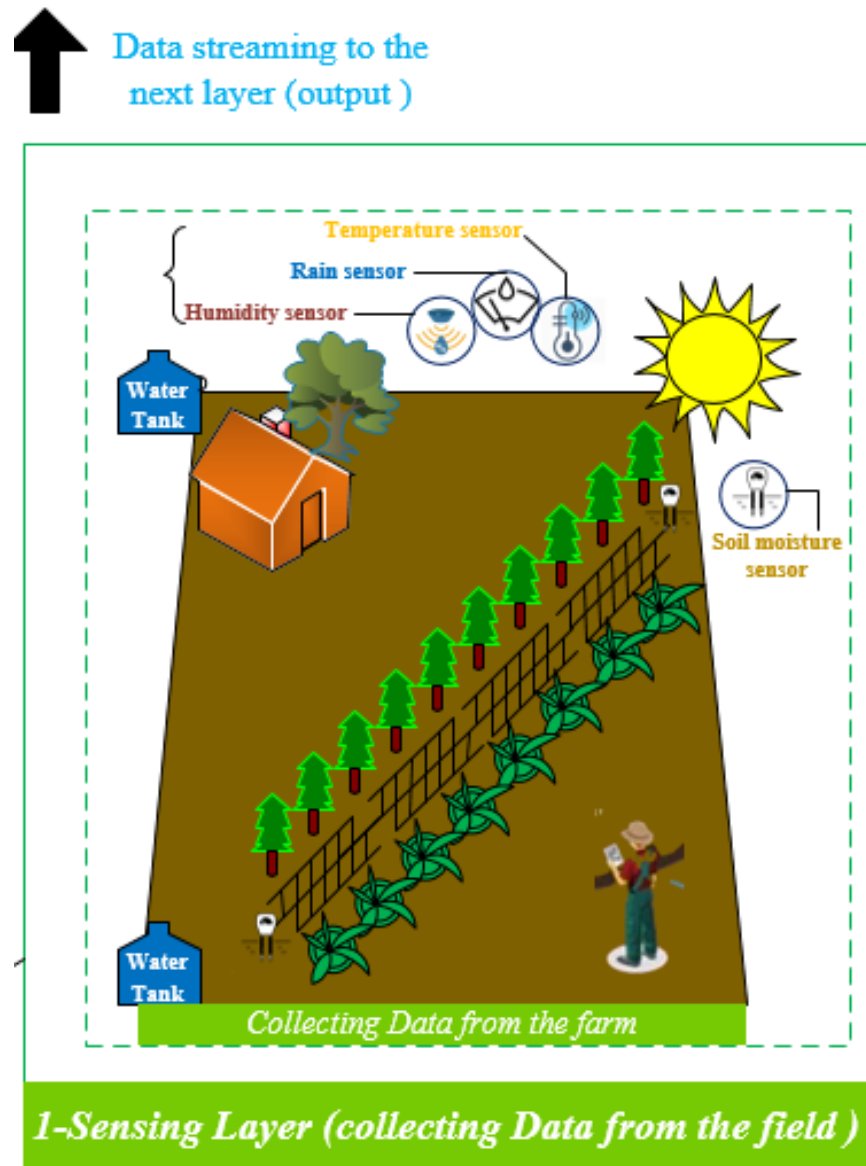


Figure 4.2: Sensing layer (collecting data from the field)

2. **Database layer :** The second layer, represented by the local server, expands its abilities and provides the platform with more functions and features that can be used in our work and it is responsible for data storage.

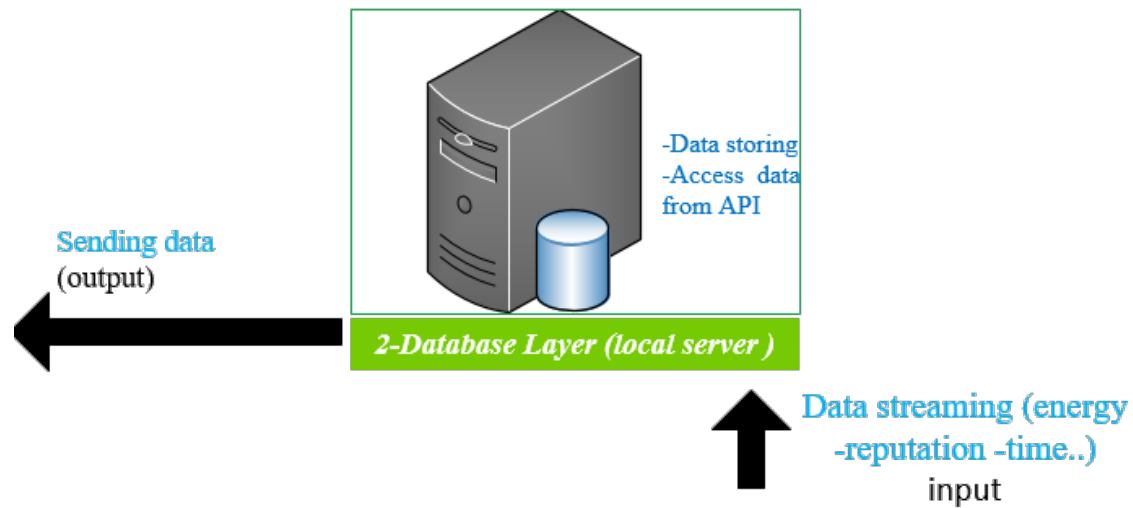


Figure 4.3: Database Layer

3. **Treatment Layer :** In the proposed architecture, this component represents the connection between the internal and external world (farmers and system) through a graphical interface. After the data collection process, all information are streamed to this layer, which is responsible for data processing. Moreover ,The main task of this device is also to capture the customer's service requirements and display the data collected after analyse it using the optimization method (evolutionary algorithms) and the combinatorial auction to determine the winner selection from a set of service compositions (the requirements model used is represented by QoS values suggested by the user and a brief description of the service).

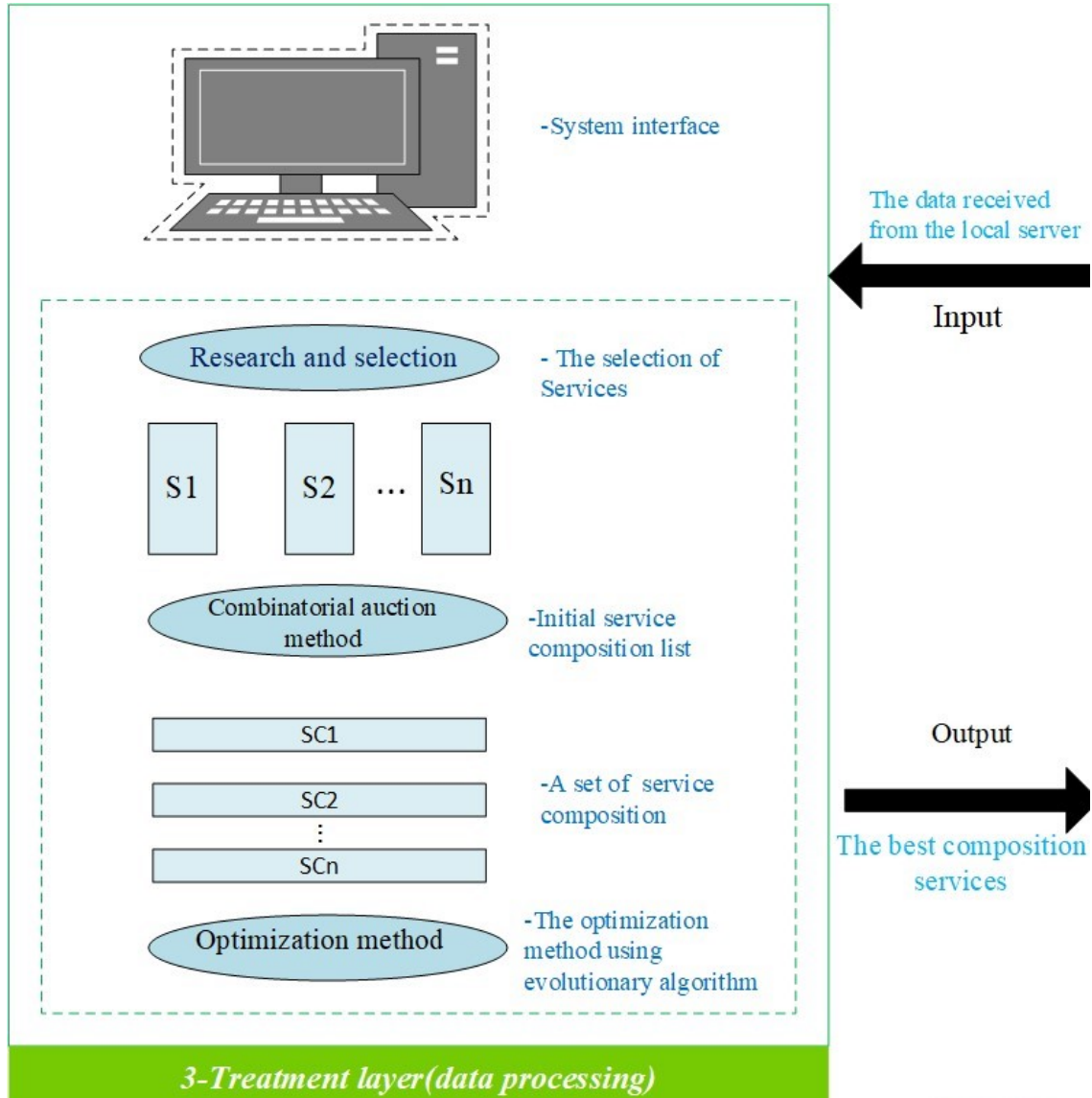


Figure 4.4: Treatment Layer

#### 4.2.2 Service composition process

The following figure 4.5 illustrates the operations involved in the proposed service composition process . It must be carried out in four steps:

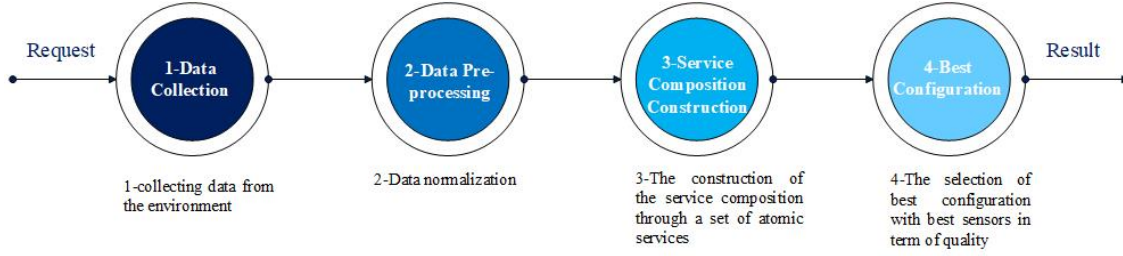


Figure 4.5: Service Composition process

1. **Data Collection:** The decision system is based on the collected data. In our case, the data are numerical values that describe the Qos values of the sensors and the ones of indicators (it shows the status of gates (distributing water in the field). The result of this process is the data collected (atomic services represented in the sensors and every sensor has it unique Qos which are the reputation, response time, distance, electricity, energy). They will be send to the local server over the streaming frame.
2. **Data pre-processing:** Collecting large amounts of small numeric values will result an incomplete datasets. Raw data can have missing values that lead to errors if entered directly into the system monitoring. Therefore, a pre-processing phase is required to clean up the data. For efficiency results, the data should be normalized in the range (0, 1) subtracting the minimum and dividing by the maximum of all observations. This pre-treatment step is important to prepare the data for the algorithm.

The normalization carried out according to the following equation :

$$S.Qos_q = \frac{S.Qos_q - Min_q}{Max_q - Min_q} \quad (4.1)$$

where  $S$  represent the service,  $Qos_q$  is the  $q^{th}$  quality of service  $S$  and  $Min$  and  $Max$  are the current minimum and maximum  $Qos$  values respectively :

3. **Service Composition construction:** To create a service composition, the process must execute the following steps:
  - **Research and selection:** Before we start the optimization algorithm, firstly we should pass through the selection of atomic services step which is in fact a greedy method.

Actually, this greedy method is maintained by using the combinatorial auction based method. The idea behind using this step is to perform a possible solutions with best configuration (set of sensors). This step is based on finding the services that are most appropriate solution for the user. Therefore, the selection is based on the quality of service (Qos) available on the server and those defined in the request. Where the selection step is based on the combinatorial auction selection method (OR-Bids method), the purpose of this step is to select the atomic services to use them for service composition process. However, the OR bids is a type of combinatorial bids that allow buyers to submit multiple atomic bids For the buyers, any number of their atomic bids, which maximizes the overall social welfare, can win [Zaidi et al. \[2018\]](#).

$$V(X) = \max_{J,K \in X} (Y(J)..OR..U(K)) \quad (4.2)$$

.

Where  $Y$  and  $U$  are the atomic services,  $J$  and  $K$  are the QoS values .

- **Execute optimization algorithm :** Since we have different types of services with huge number which we can classify this problem as NP-complete. In order to explore the whole search space it will take time where the client does not have that much of time. To deal with the mentioned problem, the ideal solution is to use an evolutionary algorithm. The idea behind the choice is in it's definition which look for optimum local (local solution in population). Moreover, our problem consists of a set of objective, to ensure the best solution with the QoS (to be maximized or minimized). The used algorithm is an evolutionary algorithms where developed because the classical direct and gradient-based techniques have problems when leading with non-linearities and complex interactions. Moreover, this algorithm uses an elitist principle , also it uses an explicit diversity preserving mechanism (Crowding distance ) and it emphasizes the non-dominated solutions. After generating the QoS values, the used algorithm tries to generate solutions that are presented over different generations in order to improve the results. When the algorithm gets the best results according to the requested, it sends the latest results with its evaluation to the



next step. The following figure 5.10 shows the steps of the optimization algorithm.

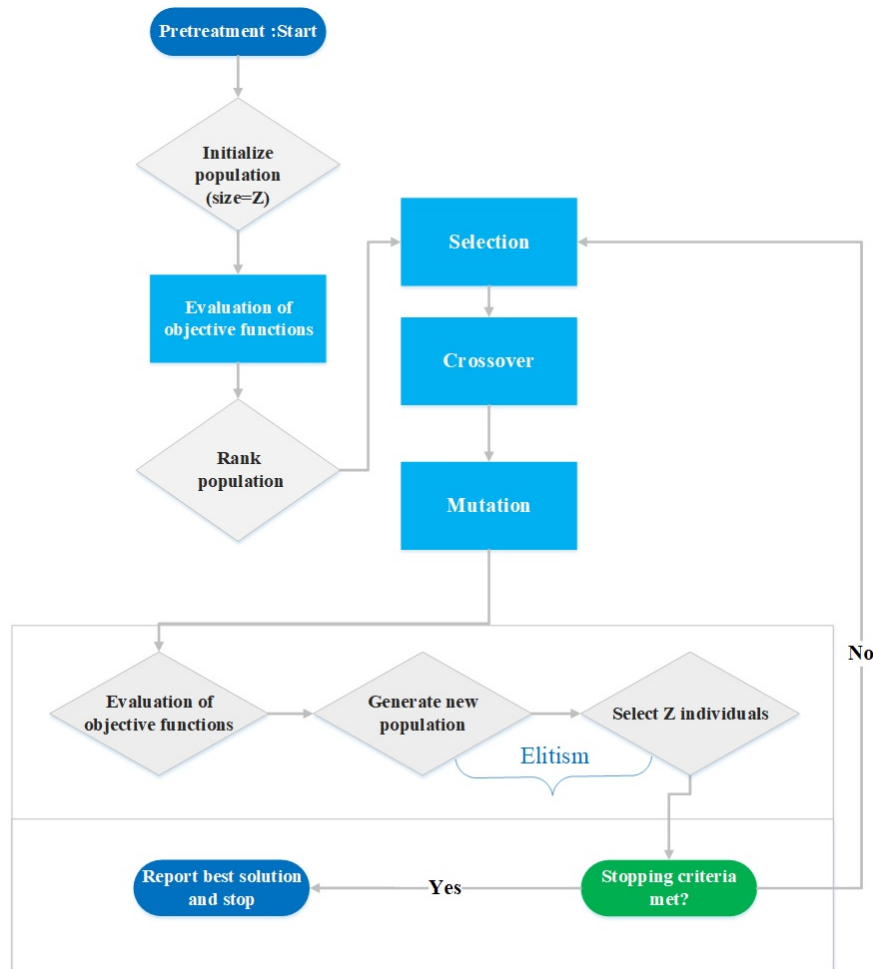


Figure 4.6: Flowchart of NSGA-II algorithm

As we have seen in the previous figure (5.10), the key steps of NSGA-II algorithm are as follows:

- (a) **The possible solution (Genome):** After generating the QoS for each service, we can construct a vector for each type of service. In the existing literature, the NSGA-II algorithm is defined as an optimization algorithm and it is one of the first multi-objective algorithms to introduce elitism, as mentioned above. The solution used in this algorithm is defined by a set of sensors represented by QoS values plus the indicators that shows the status of the water gate. The next figure shows a genome in our population:

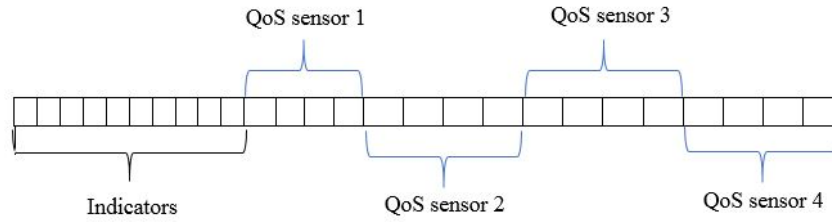


Figure 4.7: The proposed genome - individual-

- (b) **Initialize population :** The initial population of NSGA-II algorithm can be selected depends on the dimension of variables or the number of indicators (which is defer from field to another). In this step we use the data collected from the local server that corresponds the services to construct the above probable solution given in figure 4.7. After that we create  $N$  solutions to start the optimization algorithm (Initialize a parent population of size  $N$  randomly,  $P_t =$ ). Each solution in the population consists of QoS values for each type of sensors and the numbers of indicators.
- (c) **Evaluation of objective functions :** The individuals are classified taking into account the fitness and the crowding distances. Fitness is an individual's ability to survive and reproduce for the next generation. we rank the individuals based on objective functions represented in 4.3. So, we selected some values to be maximized and others to be minimized. We have chosen response time, energy, electricity, and distance to minimize and maximize reputation.

$$Fitness = \sum [\min(I + RT + D + E) + \max(R)] \quad (4.3)$$

Where  $RT, R, E, D$  represents respectively response time, reputation, the sensor's energy and distance between the sensor and the indicator of the sensors used in the solution.  $I$  represent percentage of consumed electricity of the indicators which is calculated as follows 4.4.

$$I = \frac{\sum_{i=0}^n I_i}{n} \quad (4.4)$$

Where  $n$  the number of indicator installed on the field and  $I_i$  is the  $i^{th}$  indicator in the solution.

(d) **Define the Pareto front :** It is a series of non dominated solutions, and when there are no objective that can be improved without sacrificing at least one other objective, these are selected as optimal solutions.

- **Non dominated sorting :** Non-dominated sorting is mainly used to classify decisions in population based on the principle of Pareto dominance. It plays a very important role in the selection operations of many multi-objective evolutionary algorithms. The dominance relationship defined below ,where ' $>$ ' represents the dominance relation, X and Y are services,  $x_i$  and  $y_i$  represents the QoS values. In this example, X dominates Y if the two conditions are verified (see equation 4.5).

$$\begin{aligned} X(x_1, x_2, \dots, x_n) &> Y(y_1, y_2, \dots, y_n) \\ (i) x_i &\geq y_i, \forall i = 1, 2, \dots, n \\ (ii) \exists j \in \{1, 2, \dots, n\} : x_j &> y_j \end{aligned} \quad (4.5)$$

- **Crowding distance :** Crowding distance is a measure of how close an individual is to their neighbors and is calculated as the sum of the individual distances corresponding to each goal that is to be lead corresponding objective function values and have a good diversity. The crowding distance is calculated as follows 4.6

$$d_{IR} = \sqrt{\sum_{i=1}^M \left( \frac{f_i x - R_i}{f_i^{max} - f_i^{min}} \right)^2} \quad (4.6)$$

Where  $d_{IR}$ ,  $M$ ,  $f_{imax}$ ,  $f_{imin}$  represents respectively :

- The normalization Euclidean distance from individual  $I$  to reference  $R$
- $M$  is the number of objectives
- $f_{imax}$  and  $f_{imin}$  are the population maximum and minimum objective value of  $i^{th}$

(e) **Rank population :** Assign a rank value to each individual in the population that corresponds to their degree of non-dominance 4.5 (a higher rank value usually corresponds to a higher degree of non-dominance) .

(f) **Selection operator :** Finally, we randomly select two candidates and compare them to a number of other candidates (the number is provided by the developer) using the dominance relationship (see equation 4.5) . When we finish this operation for the two selected candidates, we choose the winner (e.g. if between S1 (0.1, 0.4, 0.6, 0.33, and 0.2) and S2 (0.1, 0.5, 0.7, 0.4 and 0.2), we can see that S2 has 3 QoS better than S1, we keep it as winner).We continue this process until we reach half the size of the first population (i.e.,  $Z/2$ ) to prepare them for the next step (crossover operator).We can summarise this operation in the following steps [Zhao et al. \[2019\]](#),[Merizig et al. \[2019\]](#):

**Step 1** Begin with  $i = 1$ .

**Step 2** Pick randomly two candidates for selection  $x_1$  and  $x_2$ .

**Step 3** Pick randomly a comparison set of individuals from the population .

**Step 4** Compare each candidate  $x_1$  and  $x_2$ , against each individual in the comparison list for domination using the domination relationship.

**Step 5** If one candidate is dominated by the comparison set while the other is less dominated, then select the later for reproduction and go to Step 7, else check Step 6.

**Step 6** If neither or both candidates are dominated by the comparison set, then use niche count to choose the winner.

**Step 7** Repeat these steps until we have a half size of the first population.

(g) **Crossover operator:** In this operation we apply the crossover method, which is inspired by the biological phenomenon and used in classical genetic algorithms. In order to reproduce new solutions with existing individuals, in this project we select two individuals randomly then we select the crossover point. After that, we choose the type of services that we want to apply the crossover on and the ones who is near to the indicators. At the end of this step we obtain N solutions. The next figure 4.8 explains the crossover process.

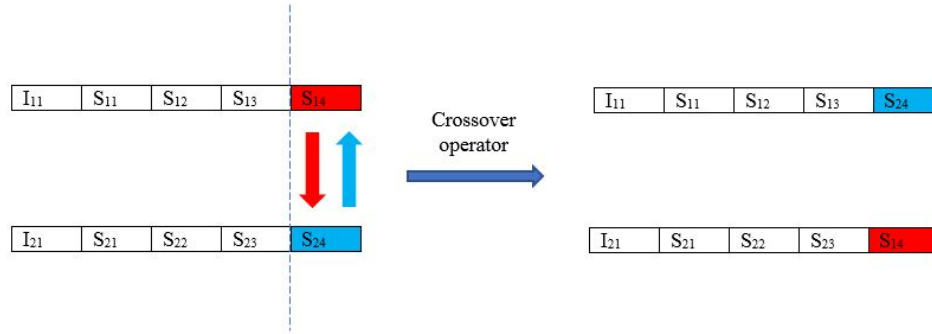


Figure 4.8: An example of crossover operation

- (h) **Mutation operation:** In the mutation operation, (we apply this process only at the indicators) we randomly exchange a gene (indicator) with another individual from the current population, taking into account the condition mentioned during the crossing operation.
- (i) **Generate new population :** Combine the parent population  $P_t$  and the offspring population  $Q_t$ , and create a new population  $R_t$  of size  $2N$ .
- (j) **Select N individuals :** Sort the parent population  $R_t$  using non-dominated method, select some individuals from the new population  $R_t$  through crowding and elitism operators, and create next generation population.
- (k) **Stopping criteria :** Jump to Step 3 and loop until stopping criterion is met (in evolutionary algorithms, there are two measures to stop the algorithm. The first one is the number of generation and the second one is that use the fitness function). In our project we used a number of generation as stopping criterion.

### 4.2.3 Used algorithm

In this subsection, we introduce the used algorithm which use the idea of combinatorial auction in the selection process for the NSGA-II to perform the service composition process. At the end of this algorithm we can provide an accurate solution or configuration for the irrigation system in a smart farm.

**Algorithm 1** NSGA-II pseudocode for CA method

---

```

1: Initialize_population_P'_of_size_N           ▷ Contains Qos values of atomic services
2: normalize_data(0,1)                         ▷ Normalizing data values in range 0 and 1
3: Combinatorial_auction_(OR_Bids)           ▷ The selection of best atomic services to
   construction the SC
4: Generate_the_offspring_population_Q'_of_size_2N
5: Evaluate_objective_values
6: Assign_rank_(level)_based_on_Pareto_sort
7: Generate_child_population_of_size_N
8: for i=0 to pop_size do
9:   for each_parent_and_child_in_N do
10:    Assign_rank
11:    Generate_set_of_non_dominated_solution
12:    Crowding_distance
13:    Crossover_and_mutation
14:    Loop_based_on_existing_solution_to_next_generation
15:   end for
16:   Select_point_on_the_lower_front_with_high_distance
17:   Generate_next_generation
18: end for
19: results ← child

```

---

### 4.3 Conclusion

In this chapter we explained our architecture and the algorithm used. our system consist in solving the problem of the composition of services due to the diversity of the services and the Qos objectives of the latter. In addition, we have proposed an architecture based on multi-objective optimization guaranteed by the NSGA-II algorithm. In the next chapter we discuss the implementation and results with some discussion.

# Chapter 5

## Implementation and results

### 5.1 Introduction

After introducing the architecture of our system in the previous chapter, we now describe the various technical aspects related to the implementation and deployment of our system. The application development process follows a series of steps, the main objective of which is to develop a product that meets the needs of customers and experts, in order to solve the problems posed and to implement and validate the ideas presented in the previous chapters.

After presenting and discussing the theoretical part of our project and the details of the algorithms and approaches used, the idea and the results obtained from it will be developed,

In this chapter we will then present the tools used including the platforms that were used to implement our system, then we indicate some interfaces of the system that provide the results obtained. At the end we give some discussion of the results.

### 5.2 Development tools and used platforms

As mentioned above, our system consists of two main parts: hardware and software. The hardware represents the physical part of the system deployed on the field, and the software represents the central system and the server, which is located in a computer or data center. In order to create a robust and efficient system, we combine some of the best advanced technologies and programming languages. The technologies used are listed below with a brief definition and explanation.

### 5.2.1 Hardware:

The hardware part mainly consists of one unit which is :

#### Sensors

For the system to work in real time, it requires sensors, which are small electronic devices that are deployed (or planted) on the field of the farm that can detect and capture information from physical objects in the area and convert it into an electrical signal . We use these devices to record various climate and soil properties that are used in the irrigation system process. The sensors used are explained as follow:

1. **Temperature Sensors:**These sensors are used to measure the amount of thermal energy in a source, the changes are then converted into data and reported to the system.
2. **Humidity Sensors:**These sensors make it possible to measure the amount of water vapor in the air atmosphere or other gases, they are placed in various devices: air conditioning (AC), heating ... etc
3. **Soil moisture Sensors:**The soil moisture sensor uses a capacitance to measure the water content of the soil (it measures the dielectric permittivity of the soil, which is a function of the water content).
4. **Rain Sensors:**A rain sensor or rain switch is a rain-activated switching device. The main use for rain sensors is as a water-saving device that is connected to an automatic irrigation system that shuts the system off when it rains.

The used sensors are listed in the following table :

Sensor	Place	Type of value
Temperature	Air	C
Hunidity	Air	%
Rain	Air	%
Soil moisture	Planted in soil	%

Table 5.1: The used sensors



## Personal computer

For running our system, we used a personal laptop with the following specs:

- **RAM:** 4.00 GB
- **CPU:** Intel(R) Core(TM) i5-4300U CPU @ 1.90GHz 2.50 GHz
- **Storage:** 238 SSD
- **System Type:** Windows 10 PRO ,64-bit operating system, x64-based processor

The QoS sensitive service selection and composition simulator was developed on the JetBrains PyCharm Community Edition 2020.3.3 platform using the Python language. The latter is a simple, portable, object-oriented language and has an extensive class library with various functions.

### 5.2.2 Software:

Our choice of programming language fell on the python language.

#### Python

For developing our system , we used Python programming language. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics [Pyt \[2021\]](#).

It's high-level data structures, dynamic typing and the huge set of libraries makes it the best choice for AI and IoT programming.

Some of the most important libraries we used:

- **Matplotlib :** is a complete library for creating static, animated, and interactive visualizations in Python.
- **NumPy :** is the fundamental package for array computing with Python.



Figure 5.1: Python

### PyCharm platform

The platform used for coding is "pycharm 2020.3.3", an integrated development environment (IDE) used in computer programming, particularly for the Python language. It is being developed by the Czech company JetBrains. It offers code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes) and supports web development with Django and data science with Anaconda [Pyc \[2021\]](#).



Figure 5.2: Pycharm

### Microsoft Visio

For painting our general architecture we used Microsoft Visio 2013 (formerly known as Microsoft Office Visio) is a vector graphics and diagram application that is part of the Microsoft Office family.



Figure 5.3: Microsoft Visio

#### 5.2.3 PyQt designer

We used PyQt for designing our system interface . PyQt is a Python connection from the Qt Cross-Platform GUI Toolkit, implemented as a Python plugin.

The most important library we used:

- **PyQt5** : Python bindings for the Qt cross platform application toolkit.



Figure 5.4: PyQt designer

### 5.3 System interfaces

This section represents the system interface component. Users interact with different interfaces to get the best service composition in term of QoS. Each interface has its own service, which is designed specifically to be easy to use and to interact with.

### 5.3.1 Home page

This component occurs the interconnection between the system and the client. It allows the user to select his requirements (sensors) regarding to quality of service (QoS). This process responsible for initialize the search and composition of the wanted service. It presents also features and services of the system, this page also provides access to other interfaces of the system.

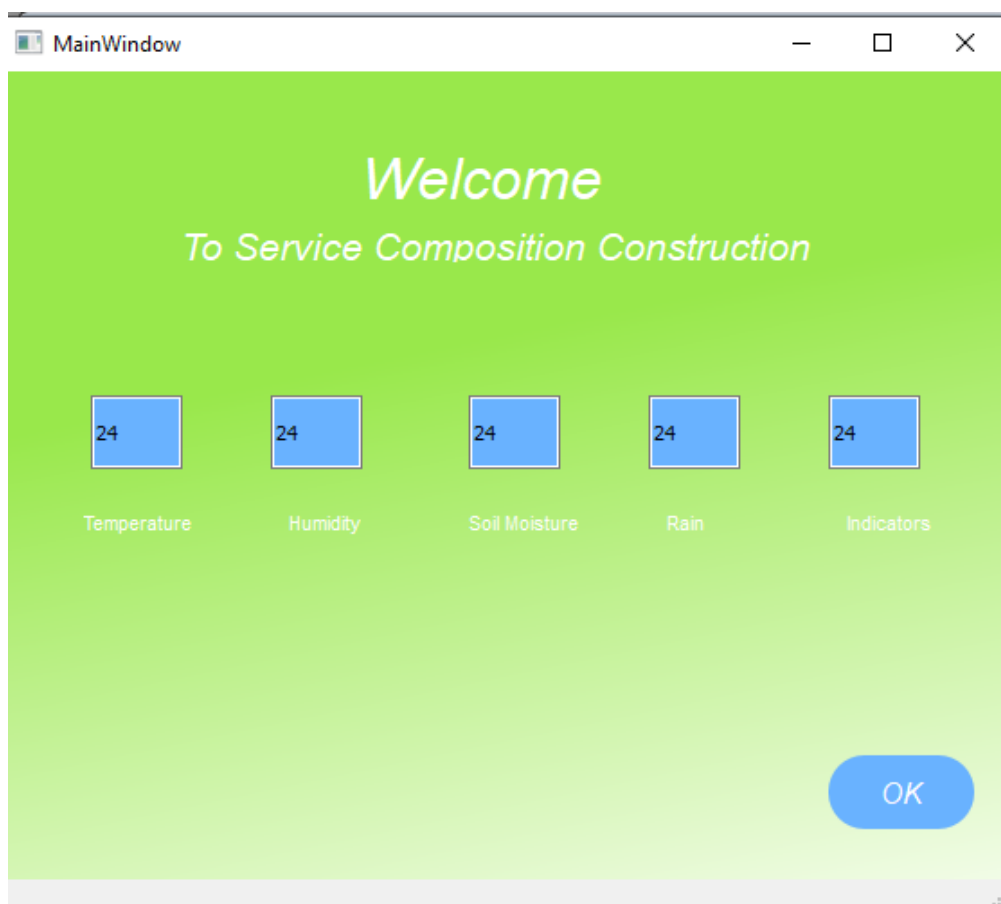


Figure 5.5: Service composition construction home page

### 5.3.2 Atomic services components

After we initialize the number of population (the number of sensors needed), the system start searching for the available atomic services. After that it displays all the services and their QoS values in order to prepare it for the next stage.

	Reputation	Response Time	Distance	<b>Energy</b>	Rep
Temperature	0.98	1.8	30	35	0.60
Rain	0.30	2.5	15	39	1
<b>Humidity</b>	0.30	2.5	15	20	1
Soil	1.3	3	15	21	0.60

[Combinatorial Auctions](#)

Figure 5.6: Atomic services components

### 5.3.3 Combinatorial Auctions selection

The system starts to select the best services in order to construct a service composition. At this step the system uses the OR-Bids methods. As mentioned above the purpose this method is to select the best atomic services to use them for service composition process. At the end of each service composition the system generates randomly the indicators. the idea behind this work is to maximize the reputation and minimize the other factors which are energy, distance and response time.

	Reputation1	Other factors1	Reputation2	Other factors2	Re
<b>Service composition 1</b>	0.70909090909091	1.47103251032511	0.63636363636362	0.34615384615384	0.727
Service composition 2	0.52727272727273	1.16694386694386	0.63636363636362	0.34615384615384	0.718
Service composition 3	0.11818181818181	0.84261954261954	0.60909090909090	1.77588357588378	0.727
Service composition 4	0.72727272727273	2.19306999306999	0.44545454545454	1.83160083160086	0.427
Service composition 5	0.11818181818181	1.04365904365938	0.26363636363636	1.8405405405404	0.181
Service composition 6	0.36363636363636	1.52633402633264	0.26363636363636	1.8405405405404	0.900

**Best Selection**

Figure 5.7: Service composition after Combinatorial auction selection

```

-----
After combinatorial auction selection
-----
***** Service Comp 1.0 *****
0.70909090909091 1.47103251032511 0.63636363636362 0.34615384615384615 0.7272727272727273 1.6417879417879417 0.7272727272727273 0.595079695079695 1.0 0.0

***** Service Comp 2.0 *****
0.5272727272727273 1.166943866943867 0.63636363636362 0.34615384615384615 0.7181818181818181 1.6255717255717257 1.0 0.7517671517671517 1.0 1.0 1.0 0.0 1.0

***** Service Comp 3.0 *****
0.1181818181818181 0.8426195426195426 0.609090909090909 1.7758835758835758 0.7272727272727273 1.7356895356895357 0.9818181818181817 1.1525294525294525 1.0 1.0

***** Service Comp 4.0 *****
0.7272727272727273 2.1930699930699933 0.4454545454545454 1.8316008316008316 0.42727272727272725 1.168953568953569 0.3636363636363636 1.307137907137907 0.0 1.0

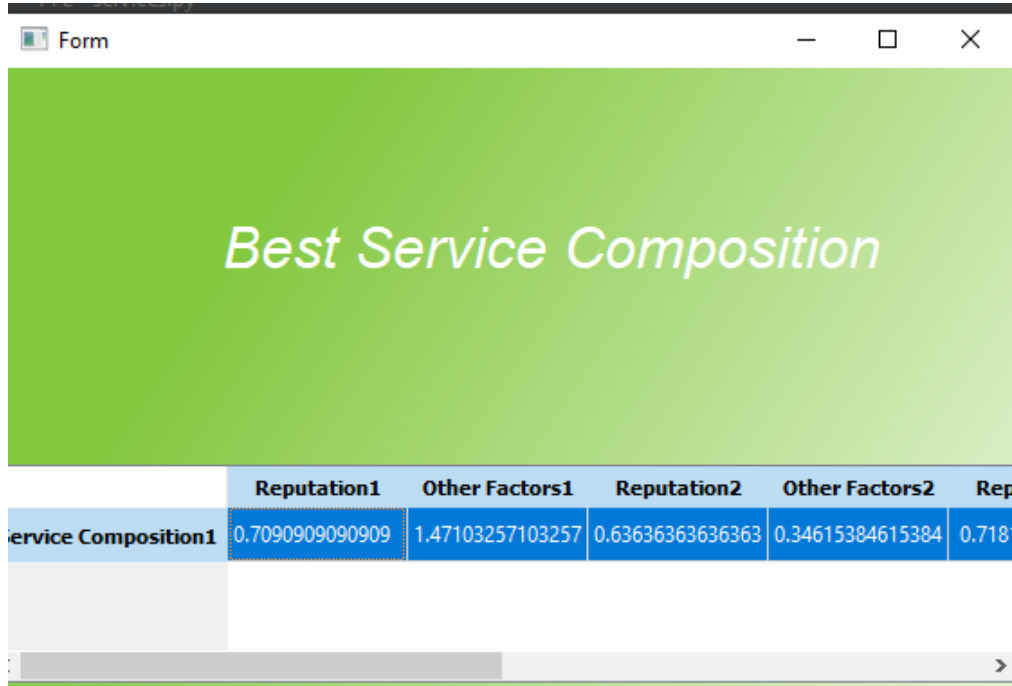
***** Service Comp 5.0 *****
0.1181818181818181 1.0436590436590438 0.2636363636363636 1.8405405405405404 0.18181818181818182 1.066666666666667 0.1727272727272727 1.6884961884961882 1.0 0

***** Service Comp 6.0 *****
0.3636363636363636 1.5263340263340264 0.2636363636363636 1.8405405405405404 0.09090909090909088 1.2166320166320166 0.0 2.3615384615384615 0.0 1.0 1.0 0.0 1.0 0
    
```

Figure 5.8: Service composition after Combinatorial auction selection

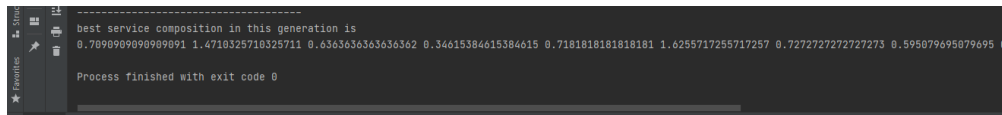
### 5.3.4 Best service composition

After constructing the composition of services the system starts to optimize the solution by using a multi-objective evolutionary algorithm NSGA-II. At the end of this step and after passing through the whole process of NSGA-II, we select the best service composition at this generation.



	Reputation1	Other Factors1	Reputation2	Other Factors2	Reputation3
Service Composition1	0.7090909090909	1.47103257103257	0.63636363636363	0.34615384615384	0.71818181818181

Figure 5.9: Best service composition



```

best service composition in this generation is
0.7090909090909 1.47103257103257 0.6363636363636 0.34615384615384 0.7181818181818 1.6255717255717257 0.7272727272727 0.59507969507969 0.7181818181818
Process finished with exit code 0

```

Figure 5.10: Best service composition

## 5.4 Obtained results and discussion

The input are the quality of services and randomly we generate the indicators. The Qos values are defined in a specific range . After that the data should be normalized for efficiency results. Service composition (output) is coded in a 1-dimensional array of two columns, represented in the Maximum of reputation and The minimum of other factors ( response time , distance , energy of the services and indicators ) and a set of indicators. The table below shows the quality of services and their range

Input	Range	Output
Reputation	[0,1]	[0,1]
Response time	[1,5]	[0,1]
Distance	[10,30]	[0,1]
Energy	[20,50]	[0,1]
indicators	[0,1]	[0,1]

Table 5.2: The used sensors

The process of the system, was carried in Pycharm platform. The process went for 100 generation with a population size of 80. the figures below shows the Pareto fronts of the first generation [5.11](#) and the fronts of the last generation [5.12](#).

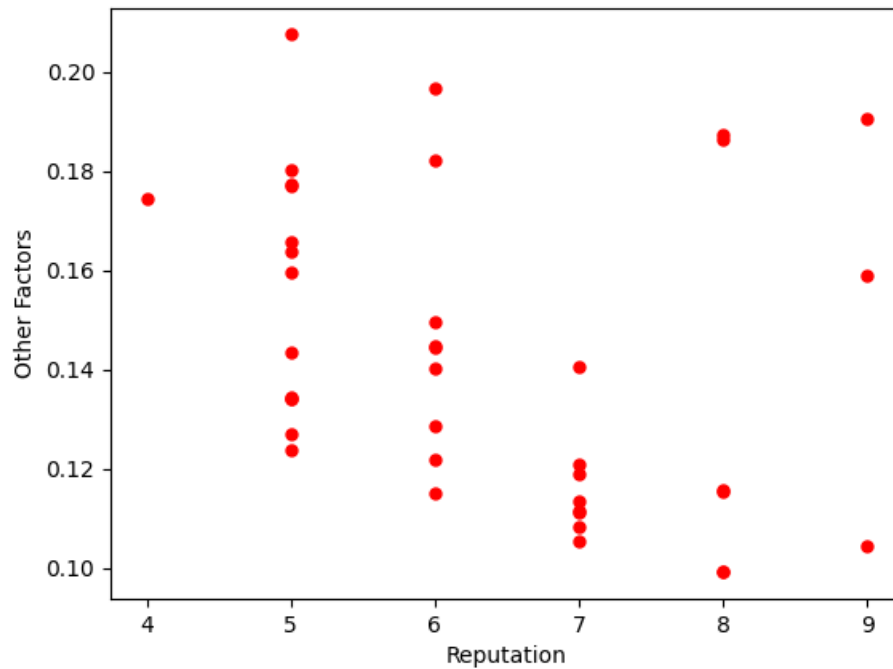


Figure 5.11: Pareto front of the first generation

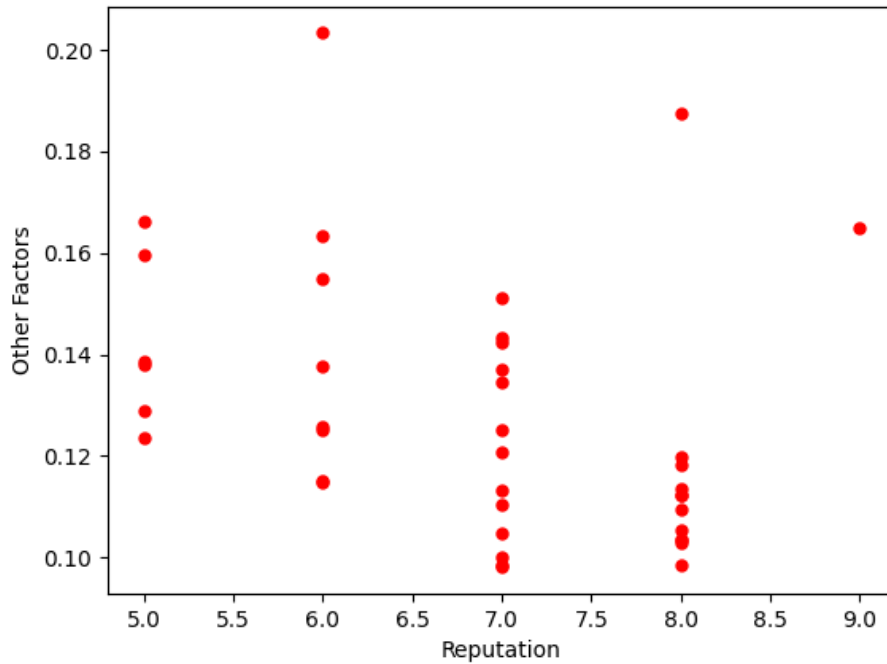


Figure 5.12: Pareto front of the last generation

At the end of this process and after selecting the best service composition in this generation, we shows up the indicators selected for supplying the farm with water in different ways. The blue colors are the open gates (1) while the yellow ones are the closed gates (0). The results of this process are shown in figures 5.13 and 5.14.

```

-----
best service composition in this generation is
0.7898989898989891 1.4718325718325711 0.6363636363636362 0.34615384615384615 0.7272727272727273 1.6417879417879417 1.0 0.7517671517671517 1.0 1.0 0.0 1.0 1.0 0

```

Figure 5.13: The final results of the status of the indicators



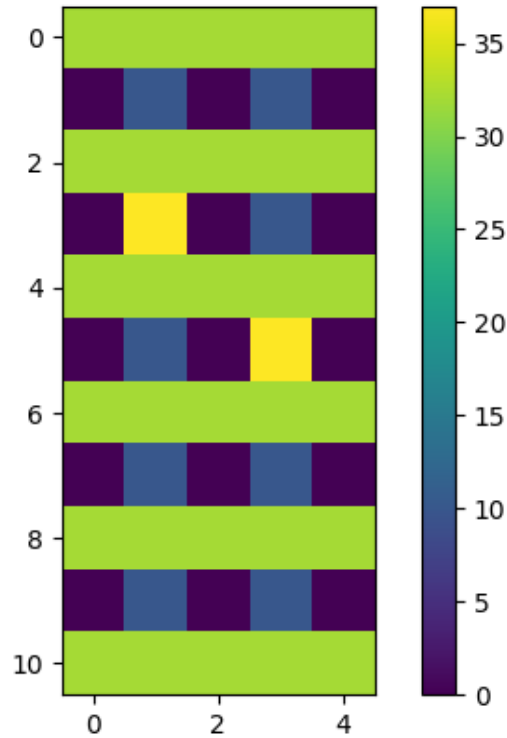


Figure 5.14: The status of the indicators

## 5.5 Conclusion

In this chapter, we introduced the implementation of our system and the results obtained. We carefully choose the best combination of tools and platforms in order to realize a service composition system with evolutionary algorithms. The results were promising and reflect a well-structured system and architecture.

# Chapter 6

## Conclusion and Perspectives

### 6.1 Conclusion

Due to the expansion of Internet services, the composition of services become a huge problem. In fact, this problem is classified as NP-complete, so it is difficult for users to find services that meet their requirements. To this end, multiple services can dynamically interact and exchange information. Service composition refers to the automatic construction of complex services based on atomic services that have gone through the selection process. In addition, the selection phase involves choosing the most suitable service to achieve a given functionality, taking into account non-functional parameters, and user needs and preferences. Therefore, to solve our proposal which is the irrigation system and to respond to the interests of users which are of different types, we had offered an approach of selection that solves this problem through the use of service composition sensitive to QoS . On the one hand, this method can choose reliable services, on the other hand, it can ensure that it meets the general requirements . Our solution presented in this project is based on the use of combinatorial auction selection approach and the optimization method.

As we have seen, problems arise when we want to provide composite services with sufficient QoS parameters. The idea behind this work was to select the best service composition using a greedy method explained in Combinatorial Auctions. Also, the use of dominance relationship encourages us to confirm the best choices among the rest through QoS settings. When you have multiple QoS to describe a service, this work turns the problem into a multi-objective problem. Using an evolutionary algorithm, we look for the local optimum without exploring the entire

search space. In addition, we have used NSGA-II, shows better performance in terms of guaranteeing the reliability of service quality objectives maximize (reputation) and minimize (energy, distance, response time).

## **6.2 Perspectives**

For future work, we aim to introduce other factors or constraints related to soil nature and quality to improve the proposed idea that consists of ensuring the irrigation in smart farms in the best condition. Plus, we plan to deploy the proposed idea in an Edge environment to ensure the data streaming step. We also plan to incorporate an analytics-based method to monitor the quality of water used in the irrigation process.

# References

Iot services. <https://www.educba.com/iot-services/?source=leftnav/>, Visited on May,20th, 2021.

The pycharm blog. <https://blog.jetbrains.com/pycharm/2019/04/collaboration-with-anaconda-inc/>, Visited on juin,20th, 2021.

What is python? <https://www.python.org/doc/essays/blurb/>, Visited on juin,20th, 2021.

Jawad Abrache, Teodor Gabriel Crainic, Michel Gendreau, and Monia Rekik. Combinatorial auctions. *Annals of Operations Research*, 153(1):131–164, 2007.

Sk Arif Ahmed, Debi Prosad Dogra, Samarjit Kar, and Partha Pratim Roy. Trajectory-based surveillance analysis: A survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(7):1985–1997, 2018.

V Anandkumar, TR Kalaiarasan, and S Balakrishnan. Iot based soil analysis and irrigation system. *International Journal of Pure and Applied Mathematics*, 119(12):1127–1134, 2018.

Liliana Antão, Rui Pinto, João Pedro Reis, and Gil Gonçalves. Requirements for testing and validating the industrial internet of things. 04 2018. doi: 10.1109/ICSTW.2018.00036.

Damian Arellanes and Kung-Kiu Lau. Evaluating iot service composition mechanisms for the scalability of iot systems. *Future Generation Computer Systems*, 108:827–848, 2020.

Thar Baker, Muhammad Asim, Hissam Tawfik, Bandar Aldawsari, and Rajkumar Buyya. An energy-aware service composition algorithm for multiple cloud-based iot applications. *Journal of Network and Computer Applications*, 89:96–108, 2017.

- Stewart Beall, Craig Carter, Phillip L Carter, Thomas Germer, Thomas Hendrick, Sandy Jap, Lutz Kaufmann, Debbie Maciejewski, Robert Monczka, and Ken Petersen. The role of reverse auctions in strategic sourcing. *CAPS research*, 2003.
- Diksha M Bhalerao and Dayanand R Ingle. Novel technique for atomic web services reliability for service oriented architecture. In *2nd International Conference on Advances in Science & Technology (ICAST)*, 2019.
- Muhammad Burhan, Rana Asif Rehman, Bilal Khan, and Byung-Seo Kim. Iot elements, layered architectures and security issues: A comprehensive survey. *Sensors*, 18(9):2796, 2018.
- Adam Chehouri. *Optimisation of wind turbine blade structures using a genetic algorithm*. PhD thesis, Université du Québec à Chicoutimi, 2018.
- Monisha Devi, Nityananda Sarma, and Sanjib K Deka. Multi-winner spectrum allocation in cognitive radio networks: A single-sided auction theoretic modelling approach with sequential bidding. *Electronics*, 10(5):602, 2021.
- Alfonso Garcia-de Prado, Guadalupe Ortiz, and Juan Boubeta-Puig. Collect: Collaborative context-aware service oriented architecture for intelligent decision-making in the internet of things. *Expert Systems with Applications*, 85:231–248, 2017.
- Matthew Gigli, Simon GM Koo, et al. Internet of things: services and applications categorization. *Adv. Internet of Things*, 1(2):27–31, 2011.
- Alan Fuad Jahwar and Adnan Mohsin Abdulazeez. Meta-heuristic algorithms for k-means clustering: A review. *PalArch's Journal of Archaeology of Egypt/Egyptology*, 17(7):12002–12020, 2020.
- Jabraeil Jamali, Bahareh Bahrami, Arash Heidari, Parisa Allahverdizadeh, and Farhad Norouzi. *Towards the Internet of Things*. Springer, 2020.
- Rongchao Jiang, Zhenchao Jin, Dawei Liu, and Dengfeng Wang. Multi-objective lightweight optimization of parameterized suspension components based on nsga-ii algorithm coupling with surrogate model. *Machines*, 9(6):107, 2021.

- Taeho Jung, Xiang-Yang Li, Lan Zhang, and He Huang. Efficient, verifiable and privacy-preserving combinatorial auction design. *arXiv preprint arXiv*, 1308, 2013.
- Diego Kiedanski, Daniel Kofman, and Ariel Orda. Design of a combinatorial double auction for local energy markets. In *International Conference on NETWORK Games, Control and Optimisation*, 2020.
- Artem Korzhenevych. Computable general equilibrium models: Historical background and basic structure. In *WORLD SCIENTIFIC REFERENCE ON NATURAL RESOURCES AND ENVIRONMENTAL POLICY IN THE ERA OF GLOBAL CHANGE: Volume 3: Computable General Equilibrium Models*, pages 3–29. World Scientific, 2017.
- Alina Lazar and RG Reynolds. Heuristic knowledge discovery for archaeological data using cultural algorithms and rough sets. In *In*. Citeseer, 2003.
- Lixing Li, Zhi Jin, Ge Li, Liwei Zheng, and Qiang Wei. Modeling and analyzing the reliability and cost of service composition in the iot: A probabilistic approach. In *2012 IEEE 19th International Conference on Web Services*, pages 584–591. IEEE, 2012.
- Mohammed Mahrach, Gara Miranda, Coromoto León, and Eduardo Segredo. Comparison between single and multi-objective evolutionary algorithms to solve the knapsack problem and the travelling salesman problem. *Mathematics*, 8(11):2018, 2020.
- James Manyika, Michael Chui, Jacques Bughin, Richard Dobbs, Peter Bisson, and Alex Marrs. *Disruptive technologies: Advances that will transform life, business, and the global economy*, volume 180. McKinsey Global Institute San Francisco, CA, 2013.
- Abdelhak Merizig. *Approche de composition de services web dans le Cloud Computing basée sur la coopération des agents*. PhD thesis, Université Mohamed Khider-Biskra, 2018.
- Abdelhak Merizig, Okba Kazar, and Maite Lopez Sanchez. A multi-agent system approach for service deployment in the cloud. *International Journal of Communication Networks and Distributed Systems*, 23(1):69–92, 2019.
- Jacob Morgan. A simple explanation of ‘the internet of things’. 2014. *Forbes Magazin*, 2017.

- Keyur K Patel, Sunil M Patel, et al. Internet of things-iot: definition, characteristics, architecture, enabling technologies, application & future challenges. *International journal of engineering science and computing*, 6(5), 2016.
- Mario Ramírez, Adolfo López, Cesareo Hernandez Iglesias, Juan Lavios, David Poza, Andrea Ranieri, Ricardo Herranz, and Adolfo Lopez-Paredes. Review of auction-based markets. 2014.
- Ammar Rayes and Samer Salam. Internet of things from hype to reality. *Springer*, 2017.
- Sherylaidah Samsuddin, Mohd Shahizan Othman, and Lizawati Mi Yusuf. A review of single and population-based metaheuristic algorithms solving multi depot vehicle routing problem. *International Journal of Software Engineering and Computer Systems*, 4(2):80–93, 2018.
- Pallavi Sethi and Smruti R Sarangi. Internet of things: architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*, 2017, 2017.
- Umar Galadima Shehu, Gregory Epiphaniou, and Ghazanfar Ali Safdar. A survey of qos-aware web service composition techniques. *International Journal of Computer Applications*, 2014.
- Seyedeh Aso Tafhiri and Saleh Yousefi. Combinatorial double auction-based resource allocation mechanism in cloud computing market. *Journal of Systems and Software*, 137:322–334, 2018.
- Bart Vangerven. *Combinatorial auctions: theory, experiments, and practice*. PhD thesis, Ghent University, 2017.
- Alexandros A Voudouris. Simple combinatorial auctions with budget constraints. *Theoretical Computer Science*, 842:6–17, 2020.
- Yu Wu. A survey on population-based meta-heuristic algorithms for motion planning of aircraft. *Swarm and Evolutionary Computation*, 62:100844, 2021.
- Mu Xia, Jan Stallaert, and Andrew B Whinston. Solving the combinatorial double auction problem. *European Journal of Operational Research*, 164(1):239–251, 2005.
- Xing Xiaojiang, Wang Jianli, and Li Mingdong. Services and key technologies of the internet of things. *ZTE Communications*, 8(2):26–29, 2020.

- Minghai Yuan, Zhuo Zhou, Xianxian Cai, Chao Sun, and Wenbin Gu. Service composition model and method in cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, 61: 101840, 2020.
- Aminu Yusuf, Nevra Bayhan, Hasan Tiryaki, Bejan Hamawandi, Muhammet S Toprak, and Sedat Ballikaya. Multi-objective optimization of concentrated photovoltaic-thermoelectric hybrid system via non-dominated sorting genetic algorithm (nsga ii). *Energy Conversion and Management*, 236:114065, 2021.
- Bizzat Hussain Zaidi, Dost Muhammad Saqib Bhatti, and Ihsan Ullah. Combinatorial auctions for energy storage sharing amongst the households. *Journal of Energy Storage*, 19:291–301, 2018.
- Bizzat Hussain Zaidi, Ihsan Ullah, Musharraf Alam, Bamidele Adebisi, Atif Azad, Ali Raza Ansari, and Raheel Nawaz. Incentive based load shedding management in a microgrid using combinatorial auction with iot infrastructure. *Sensors*, 21(6):1935, 2021.
- Mingwu Zhang and Bingruolan Zhou. Pp-vca: A privacy-preserving and verifiable combinatorial auction mechanism. *Wireless Communications and Mobile Computing*, 2020, 2020.
- Hong Zhao, Zhi-Hui Zhan, Wei-Neng Chen, Xiao-Nan Luo, Tian-Long Gu, Ren-Chu Guan, Lan Huang, and Jun Zhang. An improved selection operator for multi-objective optimization. In *International Symposium on Neural Networks*, pages 379–388. Springer, 2019.
- Weifeng Zhong, Kan Xie, Yi Liu, Chao Yang, and Shengli Xie. Multi-resource allocation of shared energy storage: A distributed combinatorial auction approach. *IEEE Transactions on Smart Grid*, 11(5):4105–4115, 2020.
- Meftah Zouai, Okba Kazar, Guadalupe Ortiz Bellot, Belgacem Haba, Nadia Kabachi, and M Krishnamurthy. Ambiance intelligence approach using iot and multi-agent system. *International Journal of Distributed Systems and Technologies (IJDST)*, 10(1):37–55, 2019.



# Appendix A

## Appendix

This section represents the main code of our system with a brief explanation to the functions included

```
5
6 class nsga2():
7     def __init__(self):
8         print('Hello , welcome to my program')
9
10    def check_range(self,x):
11        if x>=0 and x<=1: #R
12            return 1
13        return 0
14        if x >= 1 and x <= 5: #Rt
15            return 1
16        return 0
17        if x >= 10 and x <= 30: #D
18            return 1
19        return 0
20        if x >= 20 and x <= 50: #E
21            return 1
22        return 0
```

Figure A.1: Check function

The check function is responsible for checking the range of the Qos values as mentioned in the previous chapter. The reputation is defined in [0,1], the response time [1,5],the distance [10,30],the energy [20,50].

```

return 0
def __input__(self, n):
    temperature = np.array([])
    rain = np.array([])
    humidity = np.array([])
    soil = np.array([])
    for i in range(0, n):
        temperature = np.append(temperature, random.randint(0, 1))
        temperature = np.append(temperature, random.randint(1, 5))
        temperature = np.append(temperature, random.randint(10, 30))
        temperature = np.append(temperature, random.randint(20, 50))
    for i in range(0, n):
        rain = np.append(rain, random.randint(0, 1))
        rain = np.append(rain, random.randint(1, 5))
        rain = np.append(rain, random.randint(10, 30))
        rain = np.append(rain, random.randint(20, 50))
    for i in range(0, n):
        humidity = np.append(humidity, random.randint(0, 1))
        humidity = np.append(humidity, random.randint(1, 5))
        humidity = np.append(humidity, random.randint(10, 30))
        humidity = np.append(humidity, random.randint(20, 50))
    for i in range(0, n):
        soil = np.append(soil, random.randint(0, 1))
        soil = np.append(soil, random.randint(1, 5))
        soil = np.append(soil, random.randint(10, 30))
        soil = np.append(soil, random.randint(20, 50))

```

Figure A.2: Input function

In the input function we defined the atomic services randomly. It represents the number of population generated by the user. After we define the number of generation and population the process starts.

After we define the number of generation and population the process starts as follows.

```

Enter the number of required generations 100
Enter the size of population 50

-----
After combinatorial auction selection
-----
===== Service Comp 1.0 =====
1.0 0.30000000000000004 1.0 0.3833333333333333 0.0 0.08333333333333334 1.0 0.23333333333333334 1.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0
===== Service Comp 2.0 =====
0.0 0.25 1.0 0.5833333333333334 1.0 0.36666666666666664 1.0 0.35 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0

```

Figure A.3: Start of the treatment

```
def norm(self, x, arr):  
    return (x-min(arr))/(max(arr)-min(arr))  
  
def sol(self, arr, population):  
    rep = np.array([])  
    rt = np.array([])  
    e = np.array([])  
    dis = np.array([])  
    for i in range(0, population.size, 4):  
        rep = np.append(rep, population[i])  
    for i in range(1, population.size, 4):  
        rt = np.append(rt, population[i])  
    for i in range(2, population.size, 4):  
        e = np.append(e, population[i])  
    for i in range(3, population.size, 4):  
        dis = np.append(dis, population[i])  
    x = (self.norm(arr[0], rep))  
    n = x  
    y = 0  
    y += (self.norm(arr[1], rt))  
    y += (self.norm(arr[2], e))  
    y += (self.norm(arr[3], dis))  
    z = y  
    x += (1/y)  
    return x, n, z
```

Figure A.4: Normalizing function

The input data is not normalized so to ensure this step we should normalize the data in range [0,1]. The function below treat the normalizing phase. While x is the value we want to normalize and max and min are the maximum and minimum values ih the range of x.

```
def services(self, arr, population):
    outp = np.array([])
    outy = np.array([])
    outz = np.array([])
    for i in range(0, arr.size, 4):
        curr = np.array([])
        for j in range(i, i+4):
            curr = np.append(curr, arr[j])
        x, y, z = self.sol(curr, population)
        outp = np.append(outp, x)
        outy = np.append(outy, y)
        outz = np.append(outz, z)
    return outp, outy, outz
```

Figure A.5: Services function

The function services returns 3 tables contain summation of the Qos values. Where outp (the sum of rep+other factors), outz (the sum of other factors which are the energy, distance, response time), outy (the sum of reputation values).

```

def CA(self, arr1, arr2, arr3, arr4):
    population = self.setPop(arr1, arr2, arr3, arr4)
    arrS_a_b = self.services(arr1, population)
    arrT_n_d = self.services(arr2, population)
    arrH_e_f = self.services(arr3, population)
    arrR_g_h = self.services(arr4, population)
    res = np.array([])
    popo = np.array([])
    for i in range(0, int(arrS.size/4)):
        x3 = max(arrS)
        c = self.where(arrS, x3)
        popo = np.append(popo, a[c])
        popo = np.append(popo, b[c])
        arrS = np.delete(arrS, c)
        a = np.delete(a, c)
        b = np.delete(b, c)

        x4 = max(arrT)
        c = self.where(arrT, x4)
        popo = np.append(popo, n[c])
        popo = np.append(popo, d[c])
        arrT = np.delete(arrT, c)
        n = np.delete(n, c)
        d = np.delete(d, c)

    x2 = max(arrH)

```

Figure A.6: Combinatorial Auctions function

```

    x2 = max(arrH)
    c = self.where(arrH, x2)
    popo = np.append(popo, e[c])
    popo = np.append(popo, f[c])
    arrH = np.delete(arrH, c)
    e = np.delete(e, c)
    f = np.delete(f, c)

    x1 = max(arrR)
    c = self.where(arrR, x1)
    popo = np.append(popo, g[c])
    popo = np.append(popo, h[c])
    arrR = np.delete(arrR, c)
    g = np.delete(g, c)
    h = np.delete(h, c)

    res = np.append(res, x3)
    res = np.append(res, x4)
    res = np.append(res, x2)
    res = np.append(res, x1)

    for k in range(0, 10):
        popo = np.append(popo, random.randint(0, 10)%2)
    return popo

```

Figure A.7: Combinatorial Auctions function

```

-----
After combinatorial auction selection
-----
===== Service Comp 1.0 =====
1.0 0.30000000000000004 1.0 0.3833333333333333 0.0 0.08333333333333334 1.0 0.23333333333333334 1.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0
===== Service Comp 2.0 =====
0.0 0.25 1.0 0.5833333333333334 1.0 0.3666666666666664 1.0 0.35 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0
===== Service Comp 3.0 =====
1.0 0.7833333333333333 1.0 0.65 1.0 0.4166666666666663 1.0 0.7 0.0 1.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0
===== Service Comp 4.0 =====
1.0 0.8333333333333333 1.0 0.7333333333333333 1.0 0.7 1.0 0.766666666666666 0.0 0.0 0.0 1.0 0.0 1.0 1.0 1.0 1.0 0.0
===== Service Comp 5.0 =====
1.0 0.8833333333333333 1.0 0.95 1.0 0.7333333333333333 1.0 0.7999999999999999 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
===== Service Comp 6.0 =====
1.0 0.9833333333333334 1.0 0.95 1.0 0.7333333333333334 1.0 0.8166666666666667 1.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 0.0 0.0
===== Service Comp 7.0 =====
1.0 1.0166666666666666 1.0 1.0333333333333334 0.0 0.4333333333333335 1.0 0.85 0.0 1.0 0.0 1.0 1.0 1.0 0.0 1.0 1.0
===== Service Comp 8.0 =====
0.0 0.5333333333333333 1.0 1.0833333333333333 1.0 0.766666666666666 1.0 0.8833333333333334 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0

```

Figure A.8: Service composition construction

The combinatorial auction gives as an input 4 arrays containing the values of the quality of services. In every time it selects the best service regarding to its Qos values (reputation + energy+distance+response time). We append the best sensor with the max Qos values and we bid the rest of services until we construct a composition of services. After that the system add randomly the indicators at the end of every service composition. This function returns a new population ready to be used for the NSGA-II algorithm.

```

def init_pop(self,n):
    temperature = np.array([])
    rain = np.array([])
    humidity = np.array([])
    soil = np.array([])
    temperature, rain, humidity, soil = self.__input__(n)
    newpop = self.CA(temperature,rain,humidity,soil)
    return newpop

def crossover1(self,newpop):
    arr = np.array([])
    for i in range(0,newpop.size):
        arr = np.append(arr,newpop[i])
    for i in range(0,int(newpop.size/18)*2):
        ccc = random.randint(1, 20)%3
        x1 = ccc * 2
        x2 = ((ccc+1)%3) * 2

        a,b = arr[i*18 + x1],arr[i*18 + x1+1]
        arr[i*18 + x1],arr[i*18 + x1+1]=arr[(i+1)*18 + x1],arr[(i+1)*18 + x1+1]
        arr[(i+1)*18 + x1],arr[(i+1)*18 + x1+1] = a,b

```

Figure A.9: The crossover function

```

-----
The Population Pt
-----
===== Service Comp 1.0 =====
1.0 1.2666666666666666 1.0 0.45 1.0 0.8 1.0 0.1666666666666666 0.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0

===== Service Comp 2.0 =====
1.0 1.0666666666666667 1.0 0.25 1.0 0.8 0.0 0.3833333333333333 1.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0

===== Service Comp 3.0 =====
1.0 1.3166666666666667 1.0 0.5833333333333333 1.0 0.85 1.0 0.7 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0

===== Service Comp 4.0 =====
1.0 1.3166666666666667 1.0 0.5166666666666666 1.0 0.8999999999999999 1.0 1.0499999999999998 0.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0 1.0

===== Service Comp 5.0 =====
1.0 1.4666666666666668 1.0 0.9833333333333333 1.0 0.9666666666666666 1.0 1.15 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0

===== Service Comp 6.0 =====

```

Figure A.10: Population Pt

The function `init-pop` represented the parent population `Pt` of size `N`. The `crossover1` is a function to duplicate the population, it reproduced a new population `Pt` to get a new population of size `2N`. We apply this phase on 2point (services) with the same type.

```
def obj_rep(self, arr):
    x = 0
    for i in range(0, arr.size, 2):
        x += arr[i]
    return x

def obj_of(self, arr):
    x = 0
    for i in range(1, arr.size, 2):
        x += arr[i]
    return x

def obj_ind(self, population):
    indc = 0
    for i in range(population.size-10, population.size):
        indc += population[i]
    return indc/10

def fitness(self, serviceComp):
    x1 = self.obj_of(serviceComp)
    x2 = self.obj_rep(serviceComp)
    x3 = self.obj_ind(serviceComp)
    return (1/(x1+x3))+x2
```

Figure A.11: Fitness function

The functions above are the objective functions of the Qos. We chose the reputation (to be maximized) and the response time, energy, distance we named them other factors (to be minimized). The last one is the objective of indicators it returns the percentage of the open gates. While the fitness returns the of sum of other factors plus indicators upturned plus the objective of the reputation.



```

def checking(self, rep1, rep2, fac1, fac2):
    if rep1 > rep2 and fac1 > fac2:
        return 1
    if rep1 > rep2 and fac1 >= fac2:
        return 1
    if rep1 >= rep2 and fac1 > fac2:
        return 1
    if rep1 < rep2 and fac1 < fac2:
        return -1
    if rep1 < rep2 and fac1 <= fac2:
        return -1
    if rep1 <= rep2 and fac1 < fac2:
        return -1
    return 0

def sortNonDominated(self, population):
    index = np.array([])
    domination = np.array([])
    for i in range(0, int(population.size/18)):
        index = np.append(index, int(i))
        domination = np.append(domination, int(i))
    repu = np.array([])
    factors = np.array([])
    for i in range(0, population.size, 18):
        subarr = np.array([])

```

Figure A.12: Non dominance sorting function

```

        subarr = np.array([])
        for j in range(i, i + 18):
            subarr = np.append(subarr, population[j])
        x = self.obj_rep(subarr)
        y = self.obj_ind(subarr)+self.obj_of(subarr)
        repu = np.append(repu, x)
        factors = np.append(factors, y)

    for i in range(0, index.size - 1):
        for j in range(i+1, index.size):
            a = self.checking(repu[index[i]], repu[index[j]], factors[index[i]], factors[index[j]])
            if a == -1:
                index[index[i]] = index[j]
                domination[index[i]] = domination[j]
            if a == 0:
                domination[index[i]] = domination[j]
    return index, domination

def crowdingDistance(self, population, index, dominations):
    distance = np.array([])
    for i in range(0, dominations.size):
        distance = np.append(distance, 0)
    maxrep = 0
    minrep = 999999
    maxof = 0
nsqa2 -> crowdingDistance()

```

Figure A.13: Fitness function

This function introduces the non dominance sorting using the dominance relationship between the service composition mentioned in the previous chapter. The function checking veri-

fied the service composition by using the condition of dominance (if x dominate Y we return 1, and if y dominate x we return -1, else if non of them dominate the other we return 0). After that we use this function to sort the service composition according to their non dominance sorting. The results of this function are shown in the figure bellow.

```

1.0 1.316666666666667 1.0 0.5833333333333333 1.0 0.85 1.0 0.7 1.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0
-----
The BEST Services Compositions after domination
-----
0.0 7.0 8.0 6.0 2.0 11.0 5.0 3.0 1.0 9.0 10.0 4.0
-----
The list of DOMINATIONS
-----
0.0 11.0 9.0 11.0 9.0 11.0 10.0 10.0 9.0 10.0 10.0 9.0
-----

```

Figure A.14: Fitness function

```

maxof= 0
minof = 999999
for i in range(0, population.size, 18):
    for j in range(i*18, i*18 + 8, 2):
        maxrep = max(maxrep, population[i])
        minrep = min(minrep, population[i])
for i in range(1, population.size, 2):
    for j in range(i * 18 + 1, i * 18 + 8, 2):
        maxof = max(maxof, population[i])
        minof = min(minof, population[i])

for i in range(0, index.size-1):
    val = 1
    for j in range(i+1, index.size):
        if dominations[i]==dominations[j]:
            val += 1
        else:
            break
    if val == 1:
        distance[i] = 99999
    elif val == 2:
        distance[i] = 99999
        distance[i + 1] = 99999
    else:
        distance[i] = 99999
        distance[i + val - 1] = 99999

```

Figure A.15: Crowding distance function

```

distance[i + 1] = 99999
else:
distance[i] = 99999
distance[i + val - 1] = 99999
for k in range(i+1,val-1):
    repa=0
    ofa=0
    repc=0
    ofc=0
    for s in range(index[k-1]*18,index[k-1]*18 + 8,2):
        repa += population[int(s)]
    for s in range(index[k-1]*18 + 1,index[k-1]*18 + 8,2):
        ofa += population[s]
    for s in range(index[k+1]*18,index[k+1]*18 + 8,2):
        repc += population[s]
    for s in range(index[k+1]*18 + 1,index[k+1]*18 + 8,2):
        ofc += population[s]
    distance[k] = math.sqrt(math.pow(((repa-repc)/(maxrep-minrep)),2)+ math.pow(((ofa-ofc)/(maxof-minof)),2))
for k in range(i,i+val-1):
    for kk in range(i,i+val):
        if distance[k]<distance[kk]:
            distance[k],distance[kk] = distance[kk],distance[k]
            index[k],index[kk] = index[kk],index[k]
return index

```

Figure A.16: Crowding distance function

The next step represents the Crowding distance (see equation 4.6. The system start to search for the max of reputation and min of other factors in order to use it in the equation. After that we put a condition in the case if we find two SC at the same front we choose both of them randomly ( because they don't have neighbors ). In the case we find more than two SC at the same front, we apply the crowding distance equation in order to select the best candidates ( the best SC has the higher crowding distance ).

```

def fronts(self, pop):
    newrepu = np.array([])
    newfactors = np.array([])
    for i in range(0, pop.size, 18):
        c = np.array([])
        for j in range(i, i+18):
            c = np.append(c, pop[j])
        newrepu = np.append(newrepu, self.obj_rep(c))
        newfactors = np.append(newfactors, (1/(self.obj_of(c)+self.obj_ind(c))))

    plt.scatter(newrepu, newfactors, c='r', marker='o', s=25)
    plt.xlabel('Reputation')
    plt.ylabel('Other Factors')
    plt.show()

def prepareGen(self, index, pop):
    newpop = np.array([])
    for i in range(0, int(index.size / 2)):
        for j in range(int(index[i]*18), int(index[i]*18 + 18)):
            newpop = np.append(newpop, pop[int(j)])
    return newpop

def mutation(self, population):
    for i in range(8, population.size, 18):
        x = random.randint(0, 9) % 3

```

Figure A.17: Pareto front function

After sorting the front with the non dominance relationship we plot the fronts obtained in every generation. Then we prepare our population for the next step. The figure bellow present the Pareto front.

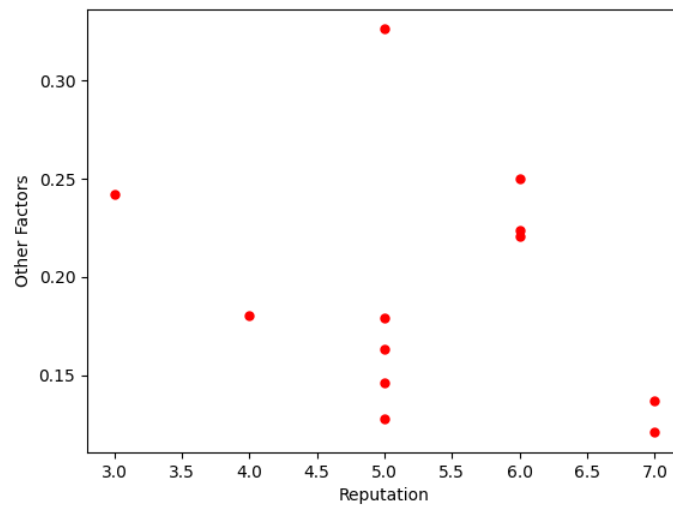


Figure A.18: The main

```

def mutation(self, population):
    for i in range(8, population.size, 18):
        x = random.randint(0, 9) % 3
        if x == 2:
            for j in range(i + x * 3, i + 10):
                if population[i] == 0:
                    population[i] = 1
                else:
                    population[i] = 0
            else:
                for j in range(i + x * 3, i + (x + 1) * 3):
                    if population[i] == 0:
                        population[i] = 1
                    else:
                        population[i] = 0
    return population

def crossover2(self, newpop):
    arr = np.array([])
    for i in range(0, newpop.size):
        arr = np.append(arr, newpop[i])
    for i in range(0, int(newpop.size/18), 2):
        ccc = random.randint(1, 20) % 3
        x1 = ccc * 2
        x2 = ((ccc + 1) % 3) * 2

```

Figure A.19: Mutation function

We applied the mutation phase only on the indicators. We devised the size of the indicators on 3 ranges randomly and we mutate them .The results of the mutation are in the figure bellow:

```

-----
The Population after MUTATION
-----
===== Service Comp 1.0 =====
1.0 1.3833333333333333 1.0 1.8499999999999999 1.0 0.9666666666666666 1.0 1.3166666666666667 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0

===== Service Comp 2.0 =====
1.0 1.3166666666666667 1.0 0.5166666666666666 1.0 0.8999999999999999 1.0 1.0499999999999998 1.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0

===== Service Comp 3.0 =====
1.0 1.3166666666666667 1.0 0.5833333333333333 1.0 0.85 1.0 0.7 1.0 1.0 0.0 0.0 1.0 1.0 0.0 1.0 1.0

===== Service Comp 4.0 =====
1.0 1.3166666666666667 1.0 0.5833333333333333 1.0 0.8999999999999999 1.0 1.0499999999999998 1.0 0.0 0.0 1.0 0.0 0.0 1.0 1.0 1.0

===== Service Comp 5.0 =====
1.0 1.2666666666666666 1.0 0.45 1.0 0.8 0.0 0.3833333333333333 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0

===== Service Comp 6.0 =====
1.0 1.3833333333333333 1.0 0.9833333333333333 1.0 0.9666666666666666 1.0 1.15 1.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0

```

Figure A.20: Mutation results

```

def crossover2(self, newpop):
    arr = np.array([])
    for i in range(0, newpop.size):
        arr = np.append(arr, newpop[i])
    for i in range(0, int(newpop.size/18), 2):
        ccc = random.randint(1, 20) % 3
        x1 = ccc * 2
        x2 = ((ccc + 1) % 3) * 2

        a, b = arr[i * 18 + x1], arr[i * 18 + x1 + 1]
        arr[i * 18 + x1], arr[i * 18 + x1 + 1] = arr[(i + 1) * 18 + x1], arr[(i + 1) * 18 + x1 + 1]
        arr[(i + 1) * 18 + x1], arr[(i + 1) * 18 + x1 + 1] = a, b

        a, b = arr[i * 18 + x2], arr[i * 18 + x2 + 1]
        arr[i * 18 + x2], arr[i * 18 + x2 + 1] = arr[(i + 1) * 18 + x2], arr[(i + 1) * 18 + x2 + 1]
        arr[(i + 1) * 18 + x2], arr[(i + 1) * 18 + x2 + 1] = a, b
    for i in range(8, newpop.size, 36):
        x = random.randint(0, 9) % 3
        if x == 2:
            for j in range(i + x * 3, i + 10):
                arr[i], arr[i+18] = arr[i+18], arr[i]
            else:
                for j in range(i + x * 3, i + (x + 1) * 3):
                    arr[i], arr[i+18] = arr[i+18], arr[i]
    for i in range(0, newpop.size):
        newpop = np.append(newpop, arr[i])

```

Figure A.21: Crossover function

```

The Off_Spring Population
-----
===== Service Comp 1.0 =====
1.0 0.3000000000000004 1.0 0.3833333333333333 0.0 0.08333333333333334 1.0 0.23333333333333334 1.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0

===== Service Comp 2.0 =====
0.0 0.25 1.0 0.5833333333333334 1.0 0.3666666666666666 1.0 0.35 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0

===== Service Comp 3.0 =====
1.0 0.7833333333333333 1.0 0.65 1.0 0.4166666666666663 1.0 0.7 0.0 1.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0

===== Service Comp 4.0 =====
1.0 0.8333333333333333 1.0 0.7333333333333333 1.0 0.7 1.0 0.7666666666666666 0.0 0.0 0.0 1.0 0.0 1.0 1.0 1.0 1.0 0.0 |

===== Service Comp 5.0 =====
1.0 0.8833333333333333 1.0 0.95 1.0 0.7333333333333333 1.0 0.7999999999999999 1.0 1.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0

===== Service Comp 6.0 =====
1.0 0.9833333333333334 1.0 0.95 1.0 0.7333333333333334 1.0 0.8166666666666667 1.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0 0.0 0.0

===== Service Comp 7.0 =====
1.0 1.0166666666666666 1.0 1.0333333333333334 0.0 0.4333333333333335 1.0 0.85 0.0 1.0 0.0 1.0 1.0 1.0 1.0 0.0 1.0 1.0

===== Service Comp 8.0 =====
0.0 0.5333333333333333 1.0 1.0833333333333333 1.0 0.7666666666666666 1.0 0.8833333333333334 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0

===== Service Comp 9.0 =====

```

Figure A.22: The offspring population function

The crossover2 consists on duplicating both of the service composition ( the same way of crossover1) and the indicators ( we devised the size of the indicators on 3 ranges).

```

pop = nsga2()
x = input('Enter the number of required generations ')
n = input('Enter the size of population ')

population = np.array([])
population = pop.init_pop(int(n))
print('\n')
print('-----')
print('After combinatorial auction selection')
print('-----')
for i in range(0, population.size, 18):
    print("===== Service Comp ", end=' ')
    print((i/18)+1, end=' ')
    print('=====')
    for j in range(i, i + 18):
        print(population[j], end=' ')
    print('\n')

for g in range(0, int(x)):
    newpop = pop.crossover1(population)
    print('-----')
    print('The Population PT')
    print('-----')
    for i in range(18*6, newpop.size, 18):
        print("===== Service Comp ", end=' ')
        print((i / 18) -5, end=' ')

```

Figure A.23: The main

```

print("==== Service Comp ", end=' ')
print((i / 18) - 5, end=' ')
print('====')
for j in range(i, i + 18):
    print(newpop[j], end=' ')
print('\n')

print('-----')
print('The Off_Spring Population')
print('-----')
for i in range(0, newpop.size, 18):
    print("==== Service Comp ", end=' ')
    print((i / 18) + 1, end=' ')
    print('====')
    for j in range(i, i + 18):
        print(newpop[j], end=' ')
    print('\n')
index, domin = pop.sortNonDominated(newpop)
newind = pop.crowdingDistance(newpop, index, domin)
preppop = pop.prepareGen(newind, newpop)
popafMu = pop.mutation(preppop)
print('-----')
print('The Population after MUTATION')
print('-----')
for i in range(0, popafMu.size, 18):
    print("==== Service Comp ", end=' ')

```

Figure A.24: The main

```

print('The Population after MUTATION')
print('-----')
for i in range(0, popafMu.size, 18):
    print("==== Service Comp ", end=' ')
    print((i / 18) + 1, end=' ')
    print('====')
    for j in range(i, i + 18):
        print(popafMu[j], end=' ')
    print('\n')
lastgen = pop.crossover2(popafMu)
pop.fronts(lastgen)
print('-----')
print('The Population PT')
print('-----')
for i in range(18 * 6, lastgen.size, 18):
    print("==== Service Comp ", end=' ')
    print((i / 18) - 4, end=' ')
    print('====')
    for j in range(i, i + 18):
        print(lastgen[j], end=' ')
    print('\n')

print('-----')
print('The Off_Spring Population')
print('-----')
for i in range(0, lastgen.size, 18):

```

Figure A.25: The main



```

print('-----')
for i in range(0, lastgen.size, 18):
    print("===== Service Comp ", end=' ')
    print((i / 18) + 1, end=' ')
    print('-----')
    for j in range(i, i + 18):
        print(lastgen[j], end=' ')
    print('\n')
lastind, lastdomin = pop.sortNonDominated(lastgen)
lastnewind = pop.crowdingDistance(lastgen, lastind, lastdomin)
print('-----')
print('The BEST Services Compositions after domination')
print('-----')
for i in lastind:
    print(i, end=' ')
print('\n')
print('-----')
print('The list of DOMINATIONS')
print('-----')
for i in lastdomin:
    print(i, end=' ')
print('\n')

res = pop.prepareGen(lastnewind, lastgen)
print('-----')
print('best service composition in this generation is ')
for i in range(0,int(n)):

```

Figure A.26: The main

```

res = pop.prepareGen(lastnewind, lastgen)
print('-----')
print('best service composition in this generation is ')
for i in range(0,18):
    print(res[i],end=' ')
population = res

out = np.array([])
for i in range(8,18):
    out=np.append(out,int(res[i]))

output = np.array([[32,32,32,32,32]])
rng = 0
todo = 0
for i in range(1,11):
    curr = np.array([])
    for j in range(0,5):
        if j%2 == 1 and i%2 == 1:
            if out[rng] == 0:
                curr = np.append(curr, 37)
            else:
                curr = np.append(curr, 10)
            rng += 1
        elif i%2 == 1:
            curr = np.append(curr,0)

```

Figure A.27: The main

```
for j in range(0,5):
    if j%2 == 1 and i%2 == 1:
        if out[rng] == 0:
            curr = np.append(curr, 37)
        else:
            curr = np.append(curr, 10)
        rng += 1
    elif i%2 == 1:
        curr = np.append(curr, 0)
    else:
        curr = np.append(curr, 32)
    output = np.append(output, curr, axis = 0)
print(output.size)
plt.imshow(output)
plt.colorbar()
plt.show()

#newind = pop.crowdingDistance(newpop, index, domin)
#print(index)
```

Figure A.28: The main

The previous figures represented the main of our program and the whole steps to execute it. At the end of the program we plot the status of the indicators. An example of the plot is shown in the figure below:

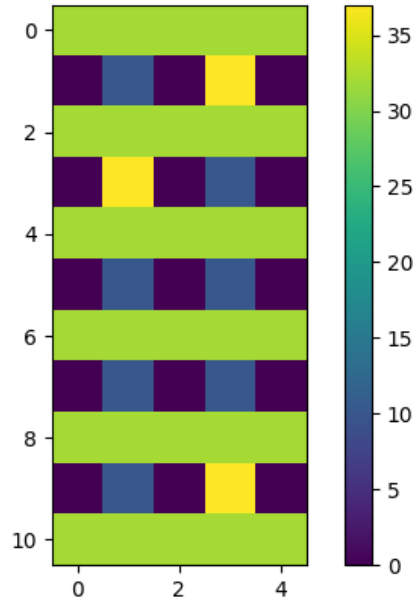


Figure A.29: The Status of the indicators

The results of our program shows the best service composition selected after ensuring all the steps of NSGA-II.

```

-----
best service composition in this generation is
1.0 1.0833333333333335 1.0 1.6333333333333333 1.0 0.8999999999999999 1.0 1.05 1.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 0.0
|
    
```

Figure A.30: An example of the last generation