electronics

MDPI

Article

Virtualization Based Efficient Service Matching and Discovery in Internet of Things

Zulfiqar Ali Khan [1], Israr Ullah [1,2], Muhammad Ibrahim [1], Muhammad Fayaz [3,*], Ayman Aljarbouh [3] and Muhammad Shuaib Qureshi [3]

1 Department of Computer Science, Virtual University of Pakistan, Lahore 54000, Pakistan; zulfiqarmrd@gmail.com (Z.A.K.); israr.ullah@vu.edu.pk (I.U.); ibrahimmayar@vu.edu.pk (M.I.)
2 Department of Computer Engineering, Jeju National University, Jeju 63243, Korea
3 Department of Computer Science, University of Central Asia, 310 Lenin Street, Naryn 722918, Kyrgyzstan; ayman.aljarbouh@ucentralasia.org (A.A.); muhammad.qureshi@ucentralasia.org (M.S.Q.)
* Correspondence: muhammad.fayaz@ucentralasia.org or fayaz@jejunu.ac.kr; Tel.: +996-770822859

Received: 9 May 2020; Accepted: 9 June 2020; Published: 16 June 2020



Abstract: Internet of Things (IoT) is getting more popular day by day, which triggers its adoption for solving domain specific problems. Cities are becoming smart by gathering the context knowledge through sensors and controlling specific parameters through actuators. Dynamically discovering and integrating different data streams from different sensors is a major challenge these days. In this paper, a service matchmaking algorithm is presented for service discovery utilizing IoT devices and services in a particular geographic area. It helps us to identify services based on a variety of parameters (location, query size and processing time, etc.). Customization of service selection and discovery are also explored. The conceptual framework is provided for the proposed model along with a matchmaking algorithm based on IoT devices virtualization. The simulation results elaborate the increased complexity of processing time with respect to the increasing pool of available services. The average processing time varies as the number of conditions are multiplied. Query size and complexity increases with additional number of filters and conditions which results in the reduction of the number of matching services. Moreover, upon decreasing the radius of geographic search area, the number of candidate services decreases for service matching algorithm. This is based on the assumption that IoT devices and services are evenly distributed in a given geographic area. Similarly, the remaining energy of IoT devices is also assumed to be uniformly distributed and, therefore, if we are interested in IoT devices or services with more residual energy, then a limited number of IoT devices or services will fulfill this criterion.

Keywords: internet of things (IoT); service oriented architecture (SOA); service provider; service matching; virtual object (VO); service object (SO); quality of service (QoS)

1. Introduction

Today with the advancement of technology and development of new standards, it is quite easy to assemble new methods in order to design and develop web based applications. The primitive web applications combine together independent web service components. These web applications are called web services [1].

The creation of business services in the form of web services is facilitated by service oriented architecture (SOA). It is an architectural paradigm that explains the IT infrastructure and makes it possible for different applications to exchange data. The whole process makes the business processes accomplish their objectives. These functions are not dependent on the underlying operating systems

and programming languages [2]. In the recent past, there has been a paradigm shift in distributed computing, from middleware oriented design to SOA.

The following entities are involved in SOA that work together to allow "find", "bind" and "invoke" operations in Figure 1.

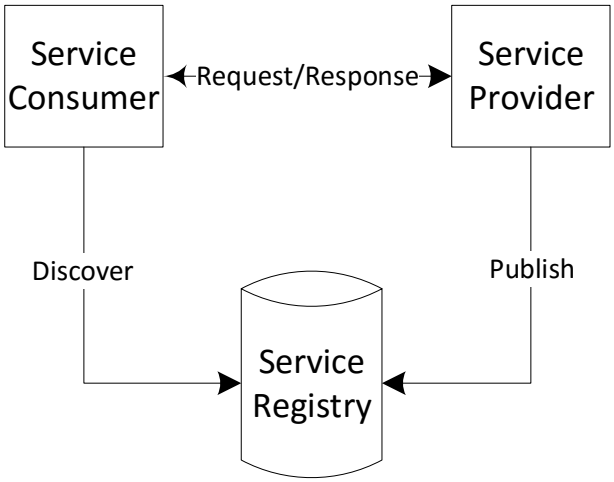


Figure 1. SOA collaborations.

Service consumer is an application or a web-service that needs another service for accomplishing its functionalities. When the request arises, it starts enquiring for the specific service in the registry. Upon successful enquiry, it binds the service over transport and service is executed according to the given requirements. Service consumer on the other hand, implements services according to the given interface. Service provider accepts request and execute it for the consumers. It provides its services and interface contract to the service registry. The services are established in such a way that the service consumer can determine and access the relevant services. Web service description language (WSDL) is a standard extensible markup language (XML) that describes web services. A service registry is used for discovering the services. Universal description discovery and integration (UDDI) is the registry used to publish web services. It has a list of all available services. It enables service consumers to use the interfaces which are provided by the service providers [3].

Web service request is initiated by the user that is followed by web service discovery and composition. The above terms are the important elements in the whole process in order to serve complicated and large service requests. Service discovery tries to match service requests with the available service advertisements and return the results accordingly [4]. Service composition on the other hand, combines several elementary web services to serve larger composite service.

Internet of Thing (IoT) devices are small by size but capable of sensing or actuating functions which are attached to daily life objects. Being limited in resources, IoT devices need storage and computing resources which could be fulfilled through cloud computing paradigm. For facilitating remote access and control, the devices can also be virtualized in the cloud environment. In the near future, the network of IoT devices will be accessed on the internet by their corresponding virtual objects (V0s). From different domains, a network of IoT devices is virtualized into VOs which will reside on cloud and multiple users can manipulate them. It could trigger sharing of resources and development of various applications on top of a virtualization layer with the help of end to end connection among IoT devices in [5].

IoT devices providing and accessing services are smaller and more resource constrained, which makes it difficult to apply the traditional approaches of service discovery. Therefore, a lightweight discovery approach is needed that works effectively with devices having restricted size, partial computational ability, limited energy and limited memory [6].

Most of the IoT devices are resource constrained that makes it necessary to have a simple matching algorithms for minimizing the processing time. At the same time, effective matching and discovery is also an important requirement. A large service request requires more resources for optimizing an efficient service discovery. This paper provides the virtualization of IoT devices as VOs. Different VOs combine to make a service object (SO). At the top, the client makes a request for any service that is handled by the service matching and discovery layer (SMDL). The discovery and matching module picks SOs by composing several VOs through a shared repository. A service matchmaking algorithm for service discovery is important in a specific geographic area. This work is the extension of our previous study presented in [7]. Previously, we proposed a services and virtual objects management (SVOM) system for efficient IoT devices and services management in hyper connected environment. There are management issues while dealing with physical devices when IoT devices and services are accessed directly. SVOM also provides virtual object-based interactive access to physical devices. However, service matching and discovery was a challenge which is addressed in this paper. In this study, we proposed an efficient service matching algorithm based on IoT devices virtualization.

The rest of the paper is structured as follows: related work is given in Section 2. The proposed model and matchmaking algorithm are presented in Section 3. The implementation and simulation are included in Section 4 along with a list of all IoT devices and their properties. The results are elaborated in Section 5 while the conclusion is given with future directions in Section 6.

2. Related Work

An independent service is called as an elementary service. Each elementary service works in a specific technical environment. However, in SOA, different services deployed over different working environments are combined to collaborate which is the main advantage of SOA. After having the capability to put together different services from different technical environments the collaborations needs to be more precise and efficient. For any service to work in the SOA paradigm, it needs to be loosely coupled, technology neutral and location transparent [8].

SOA is a suitable platform for independent decentralized systems which are autonomous. These autonomous systems help in the fulfillment of the service requests. During the service discovery process, efficient and precise matchmaking of services is required. Initially services have been discovered on the basis of syntactical matching. The matchmaking based on syntactical basis checks the service requests with the service advertisements on the basis of service descriptions. This approach is not very effective [9] and not viable for IoT environments.

Service matchmaking based on semantics performs matchmaking on the basis of semantics description. The approach seems to be more sensible and logical as compared to syntactical matchmaking. The decisions based on semantic descriptions are more effective as compared to the ones based on syntactical descriptions but currently there is no tool available that could cover the semantics of a service in all respects. However, considerable improvements have been achieved by various semantic service specification languages like Ontology Web Language (OWL-S), WSDL and SAW-OWL-S [10].

Matchmaking between service requests with service semantics on the basis of input/output parameters is also not very efficient for the generation of precise match due to the following two reasons [11]:

- Service functionality is not clearly expressed by its input-output parameters.
- The service execution is not guaranteed on the basis of input-output parameters match.

Functionality matching solves the two problems mentioned above. It is based on service functionalities and employs an approach to consider matchmaking based on input-output parameters, precondition-result situations and a variety of other service attributes. With the help of OWL-S and situation ontology, it is able to discover services with more accuracy and precision [11]. There are two phases of the given approach. First it removes the incompatible services and then selects the top service from a sorted list of compatible ones in the second phase. Thus web services which

are "similar enough" to the requested service are returned. Besides all pros of functionality match, it doesn't consider quality of service (QoS) issues while selecting web services that greatly hampers the overall performance.

Precisely matched services give the exact service that we need while efficiency is always the top priority in any type of system. Users don't want to wait for any facility over the internet rather they need to be served in time. We need to satisfy the functional requirements while service discovery and matching but we also need the non-functional requirements to be fulfilled. Most of the work has been carried out regarding functional aspects of service discovery but QoS needs further efforts to be explored [12].

Web services depend on the server machines. There are many aspects that can potentially affect the QoS such as connectivity status, service location, and device status, etc. [13]. The geographic region of servers plays an important role in the context of non-functional requirements of web service discovery and matching [14]. If a server providing a specific web service is far apart geographically then surely the response time of such service will be reasonably high. Therefore services are discovered and composed from servers which are nearest. Servers are most of the time crowded in day times while having low load in late hours and during midnight. So accessing a nearer server in day time is not always the best option as compared to a free server at a higher distance.

Wenyu et.al have proposed Levenshtein distance algorithm for matching between requested services and release of the most relevant service in [15]. The edit distance algorithm is used for computing the name similarity among the available pool of services followed by employing cosine theorem for calculating the text similarity. At the end, counting the input/output similarity is performed through semantic distance that finally filters out low matched services. The results validate the effectiveness of multi-level filtering algorithm for improving the recall and precession ratios for web services matching. However, QoS criteria are not taken into account in this research.

Service matching is very important for web service discovery and selection. The selection of most appropriate services is the main objective of service matching. At least one of the following shortcoming exists in the current matchmaking frameworks in [16]:

- Strict syntactic matching;
- Matching based on capability;
- Deficiency of customization support;
- Absence of accurate ranking of matched services.

A parameterized matching-ranking framework (PMRF) is presented in [16]. It addresses the given four problems except partial fulfilment of the second one. Among the two layers, the first one parse the input data along with parameters while the second layer deals with matching and ranking. The results show that PMRF outperforms iSeM-logic-based and SPARQLent algorithms. While matching and ranking the web services, the aspects of imprecision and uncertainty are not taken into consideration in the current framework.

Recently, in heterogeneous sensor environment, a lot of sensors are employed to collect context information for monitoring. From an engineering perspective, sensors collect data about their surroundings. Sensor middleware enables the clients to view data taken by the sensors via the internet. Chen Nan et.al has proposed an integration architecture for large scale heterogeneous sensor networks where multiple sensor middleware are being used by the service provider in [17]. The author has presented neither simulation results nor any other way to prove the effectiveness of the proposed idea.

Software and hardware communication technologies have made it easy to transform this world into a network of smart devices. These devices will not only share their sensed data but also collaborate with each other to serve larger IoT services. The beneficiary of these services will be both humans and machines utilizing sensors and actuators. The majority of the research has been focused on the connectivity and management of sensing devices but little has come up for actuating devices. A generic process model is proposed for controlling the actuating devices in future networks in [18]. The authors

have presented the implementation of the system in the form of a prototype. Performance analysis has been presented for control of service provisioning at both service and application layers. However, the prototype implementation does not have the autonomous control.

For large scale registration in IoT environment, service discovery is quite difficult. In order to match a service, all the matching parameters need to be matched if a traditional approach for service matching is used. However, it results in inefficient usage of computational resources and time. A multi-stage semantic matching discovery algorithm is presented by Bing Jia et al. in [19]. The algorithm considers category of service category, input/output, precondition vs. effect and quality of experience as matching criteria. IoT services and requests are expressed through OWL-Siot. A structured model is presented that is composed of interactive interface, parsing annotation, service matching and data semantic layers. A hybrid service matching degree measurement is also proposed in the given study. Precision and recall are improved according to the author but proper simulation has not been performed.

There is a need of common nomenclature for the description of IoT devices in order to address the issues of device interoperability. For IoT devices to have a common interface, a swarm framework has been proposed by Pablo et al in [20]. The study facilitates decentralized communication and interaction among IoT devices. Enabling semantic discovery in the swarm through open web technologies is the ultimate objective of the study. However, further research is required for a framework that is fully distributed following swarm principles.

The devices in IoT environment are heterogeneous in nature. Due to the diversity in IoT environment along with geographical distribution of services make it a challenging task to match and discover services. Techniques which make use of complex algorithms and powerful infrastructure is not feasible due to resource constraint nature of IoT devices. The author has presented a secure broker based discovery approach for services that takes care of service context and provider truth worthiness along with QoS parameters in [21]. The proposed technique achieves negligible latency and power increase without wasting resources of IoT devices. However this study is focused on the development of a secure mechanism for service discovery in IoT environment while we have proposed an efficient methodology for service discovery based on virtual objects. When the number of IoT devices and services grows then management issues arises if they are accessed and manipulated directly. While, proposed system provides an easy to manage and scalable scheme for IoT devices and services management and discovery. IoT devices are represented in system through virtual objects (VOs). These VOs are combined to compose desired IoT services. A mechanism for building a virtual network among connected IoT devices from different domains through their corresponding virtual objects in the cloud environment is presented in [5]. However, after service composition, efficient service discovery was an challenging issue which is being addressed in this paper.

3. Proposed Model

For developing IoT based applications, virtualization is the first step for abstracting properties of IoT devices into virtual objects (VO). After that, multiple VOs can be combined for composing various services for a variety of applications. Several existing systems providing IoT devices virtualization and composition of services lacks its proper management when too many devices become part of IoT network. Previously, we have developed an architecture for IoT services and virtual objects management in a hyperconnected IoT devices environment. The proposed model facilitates the management issues of IoT devices network [7]. A prototype of services and virtual objects management (SVOM) system is also implemented for effectively managing and monitoring IoT devices and services. SVOM system support interactive control operations on selected IoT devices. However, service matching and discovery was a challenge which is addressed in this paper. In this study, we have proposed an efficient service matching algorithm based on IoT devices virtualization by adding a service matching and discovery layer (SMDL) to the proposed architecture. The conceptual layering of the proposed model is presented in Figure 2.

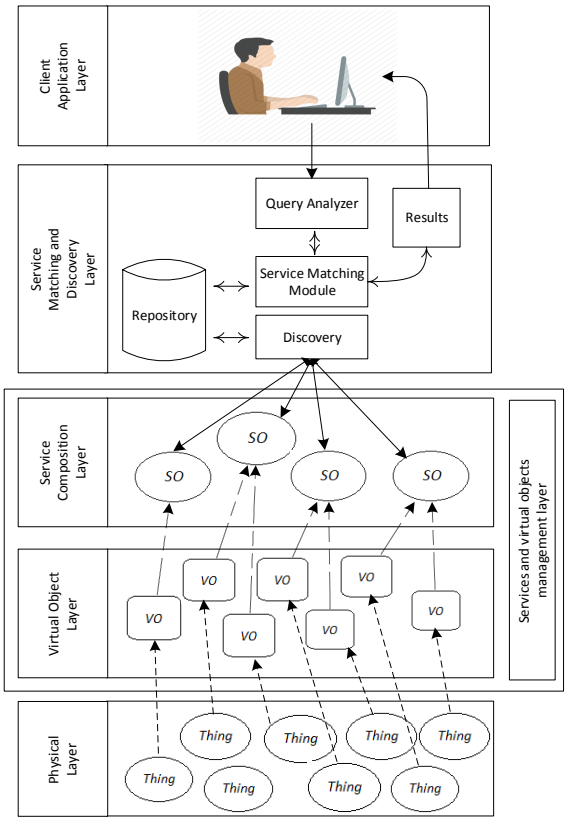


Figure 2. Conceptual layering for IoT devices and service matching based on virtualization.

At the physical layer there are actual things/objects which are exposed to the outside world. The things could either be electrical or electromechanical components for taking and controlling environmental parameters respectively. They have actuating and sensing capabilities in order to control certain parameters in a specific environment.

The virtual object (VO) layer deals with the virtualization of physical objects. Each physical thing is modeled through a VO. It encapsulates the behavior and state of any physical object. Users manipulate the physical things via their virtual representations.

Service composition layer, combines different VOs to make a service object (SO). A primitive SO receives data from an input VO and displays it through an output stream. A sophisticated SO could be connected to several sensors (pressure, temperature and humidity, etc.) that takes the input from its environment and activates the corresponding output actuators like alarm, lights or preemption of relevant services.

At the top, the client makes a request for any service that is handled by the service matching and discovery layer (SMDL). This layer first analyses the client's query followed by incorporating any matching among services. Discovery and matching module picks SOs by composing several VOs through a shared repository.

The proposed model provides efficient matchmaking for a requested service from a pool of available services as shown in Figure 3. When a new service is requested, it is matched based on its name along with location (optional). Name matching is facilitated by an option of partial or full string matching. Upon successful match, it is added to the set of candidate services. In the second phase, matching is performed based on functionality. Upon successful functionality match, the services are added to the set of compatible services list. At the end, filtering is carried out on QoS criterion. The final list of services is produced after passing through the given three filters.

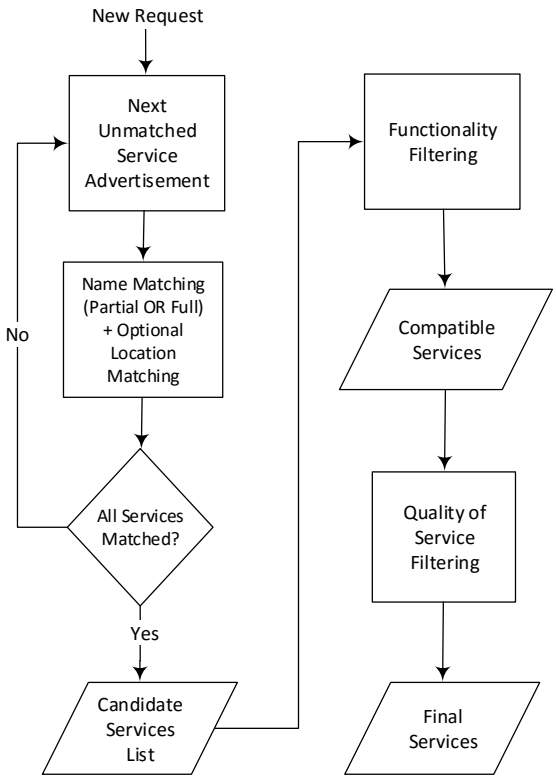


Figure 3. Proposed matching algorithm.

In essence, we have the following filtering stages for service matching,

1. Name plus location (optional) matching;
2. Functionality matching;
3. QoS matching.

Functionality matching is performed by comparing the input and output parameters of requested service with the list of available services from a candidate set. In the name matching, services will be matched based on location that will be optional but it is very handy for relevant matchmaking. In addition with exact location, a range is also provided by the user for a requested service that determines the 360 degree coverage from a specific geographical location. A threshold value is also used during matching for the difference between values of condition variables. An absolute difference between requested and available condition values is compared with threshold in the given matchmaking procedure.

The pseudo code of the above procedure is given in Algorithm 1.

Algorithm 1: Matchmaking Procedure

Input: r : requested service, s_i: ith service, m: service name matching criteria (partial or full), f: service functionality matching criteria (partial or full), τ: threshold, C : candidate services = $\emptyset$

Output: the set of final services among S

```
if m=fully then
    foreach si ∈ S do
        if s_i == r.name then
            C = C + s_i
else
    /* Partial Name Matching                                   */
    foreach si ∈ S do
        if left(s_i) = left(r) || right(s_i) = Right(r) then
            C = C + s_i
if r.loc ! = ∅ then
    C_f = empty
    foreach C_i ∈ C do
        if dist(C_i.loc, r.loc) < r.d) then
        C_f = Ci + C_f
foreach Ci ∈ C_f do
    /* Functionality Matching                                  */
    if f = partial then
        if C_i.in = r.in && C_i.con.op = r.con.op &&
          |C_i.con.val − r.con.val| ≤ τ)||C_i.out = r.out then
            C_p = C_i + C_p
        else
            if C_i.in = r.in && C_i.con.op = r.con.op && |C_i.con.val − r.con.val| ≤ τ) &&
              C_i.out = r.out then
                C_p = C_i + C_p
foreach C_i ∈ C_p do
    /* QoS Matching                                            */
    if C_i.Qos ≥ r.Qos then
    C_f = Ci + C_f
```

Where r is the requested service by the user and it can be expressed as $name, in, out, con, QoS, [loc, d]$. in is the input property of sensor, out is the output property of actuator, con has further two components i.e., op, val while op is the condition operator and val is requested condition value.

4. Implementation and Simulation

This section mentions the development and testing tools used in this study. In the proposed model, the services are implemented in C-Sharp using Microsoft Visual studio 2015 [22]. The data and service repositories are implemented in SQL Server 2014. The development environment runs over Windows 10 on Intel Core i5 processor with 8 GB memory.

A total of 4000 devices are spread across central Islamabad, the capital of Pakistan. There are 32 types of devices (sensors and actuators). The profile of each device is created as VO in SQL

that is composed of device id, URI, type, location, latitude, longitude, size of each property and total number of properties. A sample list of devices are shown on Google Map in Figure 4 over an area of 44,000 square meters. Table 1 lists down all the sensors and actuators used for sensing the environmental parameters and manipulation of the same respectively. The table also shows the corresponding properties of each device.

The experiments are conducted by virtualization of IoT objects. VOs play a significant role in order to improve the energy management efficiency of IoT objects along with solving the issues of scalability and heterogeneity. IoT devices are accessed and controlled using their corresponding VOs. This helps in maximizing overall lifetime of IoT networks and also allow to include new protocols and configuration of IoT devices. Functionality of IoT devices is captured through emulators which also helps us in replicating a number of IoT devices for conducting scalability analysis of the proposed system. A slight change in results is expected with real IoT devices due to network latency depending upon the device placement and location. Therefore, a minor deviation in results is expected while experimenting with real IoT devices. However, the performance of the proposed service matching algorithm will remain the same as it is based on virtual instances of real IoT devices.

Each device has certain properties which can be used to get data from or activate it after successful matching. For example, a garden sensor can be used to get the temperature, light, moisture and humidity of a particular location. Upon getting the required data, the corresponding actuator-like solenoid can be activated or stopped to regulate/stop the flow of water. Similarly, a smoke sensor can be installed at public places to ensure clean environment. When the smoke sensor detects the ion concentration beyond a specific limit in the air, the associated buzzer could start ringing.

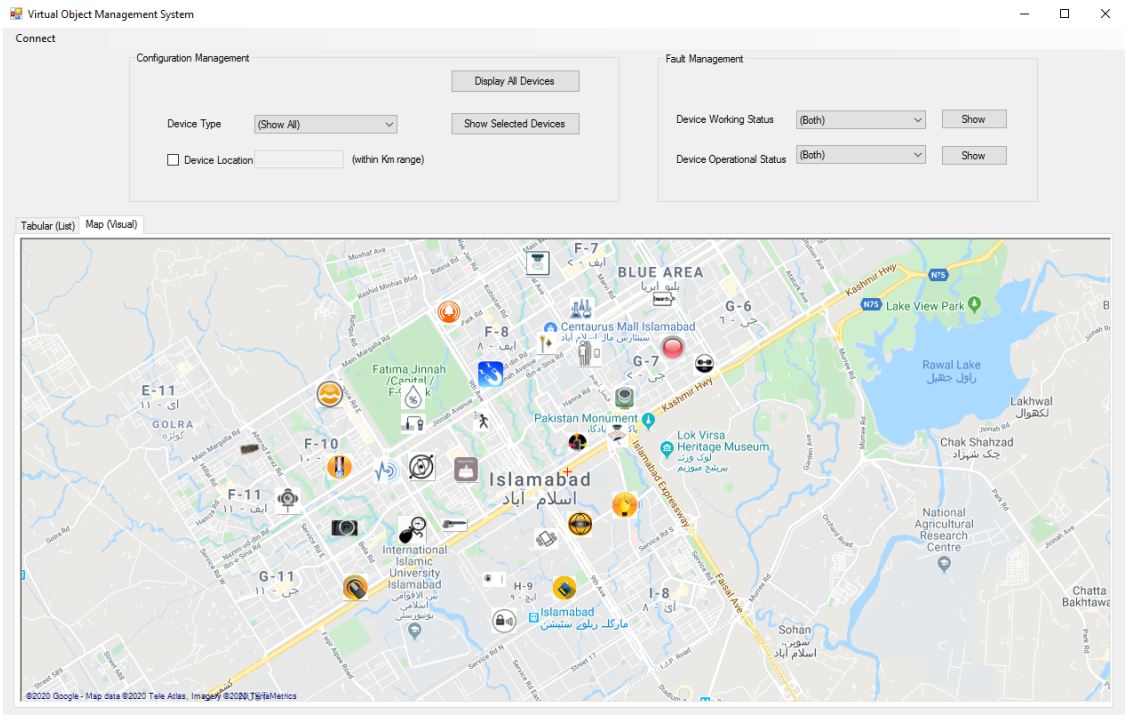


Figure 4. IoT devices on Google Maps.

Table 1. List of sensors and actuators.

Device No.	Name	Properties
1	Camera	Get Frame, Start, Stop, GetStatus
2	Temperature Sensor	GetTempC, GetTempF, GetStatus
3	Proximity Sensor	ObjectPresent, Start, Stop, GetStatus
4	Pressure Sensor	GetPressure, Start, Stop, GetStatus
5	Level Sensor	GetSwitchStatus, GetLevelReading, Start, Stop
6	Image Sensor	GetFrame, Stop, Start, GetStatus
7	Motion Sensor	GetFrame, Stop, Start, GetStatus
8	Accelerometer	GetRateofChangeofVelocity, GetChangeinMagneticField, Start, Stop, GetStatus
9	Vibration Sensor	GetResonance, Start, Stop, GetStatus
10	Rain Sensor	GetTotalInternalReflection, Start, Stop, GetStatus
11	Weight Sensor	GetWeight, Start, Stop, GetStatus
12	Axis Sensor	GetXAngle, GetYAngle, GetZangle
13	Gyroscope	GetRotationSpeed, Start, Stop, GetStatus
14	Humidity Sensor	GetWaterVapours, Start, Stop, GetStatus
15	Optical Sensor	GetLightRays, Start, Stop, GetStatus
16	Touch Sensor	GetVoltageChange, Start, Stop, GetStatus
17	Gas Sensor	GetCarbonDioxide, Start, Stop, GetStatus
18	Smoke Sensor	GetIonConcentration, Start, Stop, GetStatus
19	Infrared Sensor	GetHeat, Start, Stop, GetStatus
20	Light Sensor	GetData, GetStatus
21	Water Quality Sensor	GetIonChlorineConcentration, GetPhConcentration, Start, Stop, GetStatus
22	Chemical Sensor	GetChemicalChange, Start, Stop, GetStatus
23	Garden Sensor	GetTempC, GetTempF, GetLight, GetMoisture, GetHumidity, Start, Stop, GetStatus
24	Magnetic Sensor	GetMagneticField, Start, Stop, GetStatus
25	Servo Motor	SetSpeed, GetSpeed, Start, Stop, GetStatus
26	Buzzer	TurnON, TurnOFF, GetStatus
27	Smart Lock	Open, Close, Start, Stop, GetStatus
28	Solenoid	GetFluxDensity, Active, NotActive, Start, Stop, GetStatus
29	Stepper Motor	SetSpeed, SetPhase, GetSpeed, GetPhase, Start, Stop, GetStatus
30	Linear Actuator	MoveForward, MoveBackward, Start, Stop, GetStatus
31	Relay	Active, NotActive, GetCurrent, Start, Stop, GetStatus
32	LED	TurnON, TurnOFF, GetStatus

5. Results

A set of experiments are conducted with 32 types of IoT devices. Every IoT device is virtualized as VO. In the simulation, a virtual network is created from a set of sensors and actuators. Data are taken from sensors and corresponding actuators are activated through VOs. There are 4000 devices deployed with 10,000 overall services for an area spanning over 11 sectors of Islamabad.

5.1. Total Services Matching Versus Processing Time

For results presented in Figure 5, experiments are conducted using a simple keep alive query that checks the operational status (busy/idle) of varying number of IoT services from 2000 to 10,000. As the number of services increases, the processing time also increases as shown in Figure 5. It can be seen that the graph increases almost linearly keeping fixed query size. For any particular service matchmaking, the average processing time increases as the density of available pool of services increases. The box plot shows minimum, maximum and median values as shown in Figure 5. The service matching algorithm is executed for 2000 services that gives 10 results. Out of 10, the minimum, maximum and median values are computed. In the same way, minimum, maximum and median values can be observed for service matching from the pool of 4000, 6000, 8000 and 10,000 services respectively.

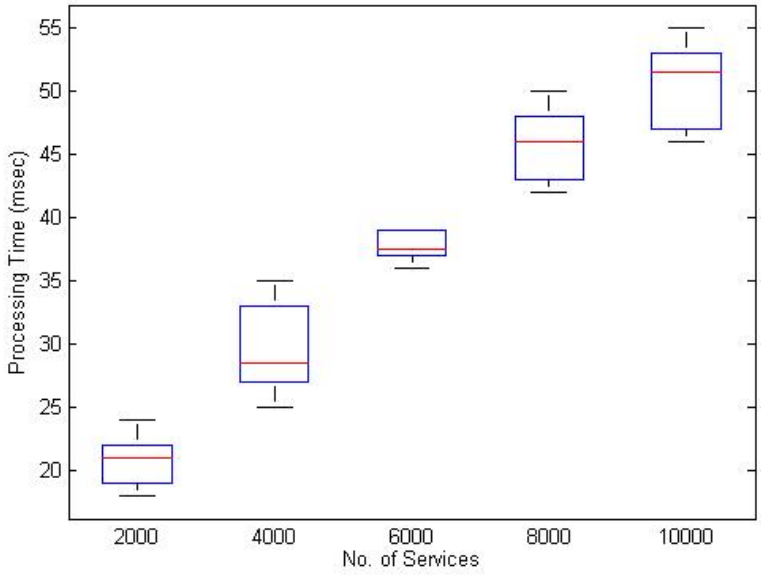


Figure 5. Services vs. processing time.

5.2. Increase in Query Size Versus Processing Time

In Figure 6, different queries (with increasing complexity) are executed against fixed number (10,000) of IoT services. Complexity of the queries is increased by adding more and more filters. For primitive queries, the execution time is lesser as compared to complex queries. That is why significant increase in the response time is observed. The light sensors measure the value of light intensity in lux. Different types of queries are used to control the amount of lighting on the road network. Qry 1 is used to find out the amount of light from all light sensors based on name matching. The input/output parameters are compared for both sensors and actuators for ultimate matchmaking. Qry 2 specifies the location to be only sector H-8 in order to be more specific. In Qry 3, the threshold value of lux is taken into account that triggers the LEDs to be ON or OFF. The light sensors compare the computed lux value with a threshold. If lux < 1, LEDs on the road will turn ON otherwise they remain OFF. The level of QoS for each device is maintained manually that ranges from 1 to 10. The acceptable QoS range is 6 to 7. Table 2 lists down the structure of different types of queries used. The variation in queries successively consume more time to be executed as shown in Figure 6. The query size is increased by adding more conditions for precision. Upon increasing the size of different queries, the corresponding processing times increases.

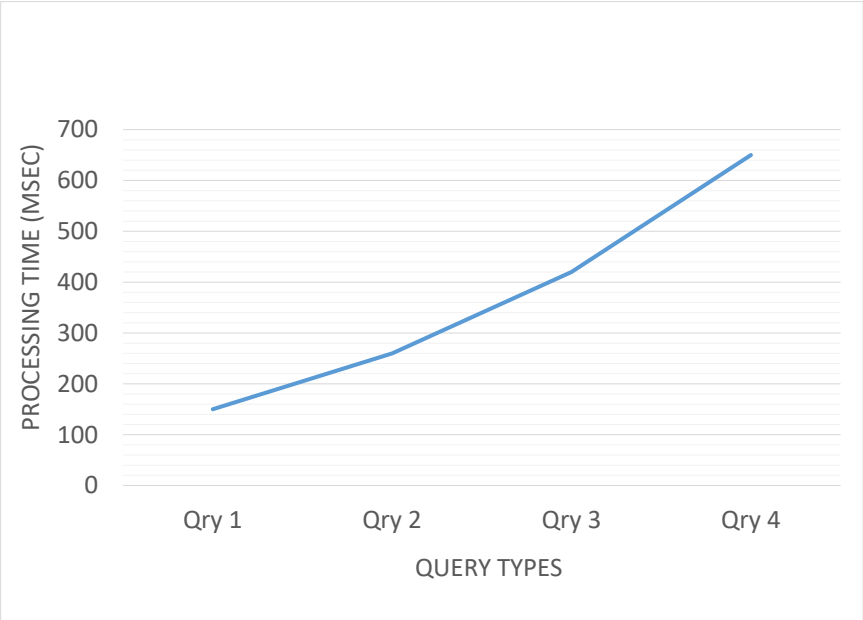


Figure 6. Variation in query types vs. processing time.

Table 2. Types of queries.

Query Name	Structure	
Qry1	$GetServices\ if\ s_i.name = r.name$	$\forall\ si \in \mathcal{S}$
Qry2	$GetServices\ if\ s_i.name = r.name$ $GetServices\ if\ s_i.loc = r.loc$	$\forall\ si \in \mathcal{S}$ $\forall\ si \in \mathcal{S}$
Qry3	$GetServices\ if\ s_i.name = r.name$ $GetServices\ if\ s_i.loc = r.loc$ $GetServices\ if\ C_i.in = r.in\ \&\&\ C_i.con.op = r.con.op\ \&\&\ \lvert C_i.con.val - r.con.val \rvert \leq \tau) \lvert\rvert C_i.out = r.out\ OR\ C_i.in = r.in\ \&\&\ C_i.con.op = r.con.op\ \&\&\ \lvert C_i.con.val - r.con.val \rvert \leq \tau)\ \&\&\ C_i.out = r.out$	$\forall\ si \in \mathcal{S}$ $\forall\ si \in \mathcal{S}$ $\forall\ Ci \in \mathcal{C}_f$
Qry4	$GetServices\ if\ s_i.name = r.name$ $GetServices\ if\ s_i.loc = r.loc$ $GetServices\ if\ C_i.in = r.in\ \&\&\ C_i.con.op = r.con.op\ \&\&\ \lvert C_i.con.val - r.con.val \leq \tau) \lvert\rvert C_i.out = r.out\ OR\ C_i.in = r.in\ \&\&\ C_i.con.op = r.con.op\ \&\&\ \lvert C_i.con.val - r.con.val \rvert \leq \tau)\ \&\&\ C_i.out = r.out$ $GetServices\ if\ C_i.QoS = r.QoS$	$\forall\ si \in \mathcal{S}$ $\forall\ si \in \mathcal{S}$ $\forall\ Ci \in \mathcal{C}_p$ $\forall C_i \in \mathcal{S}$

5.3. No. of Matched Services after Each Filtering

The three matching criteria are given as follows,

1. Name plus location (optional) matching;
2. Functionality matching;
3. QoS matching.

The number of matched services will decline with any increase in query size for service matching in Figure 7. Increasing the number of filters shows a declining graph. This is due to the increasing complexity of the proposed algorithm which yields lesser services as successive filters are applied.

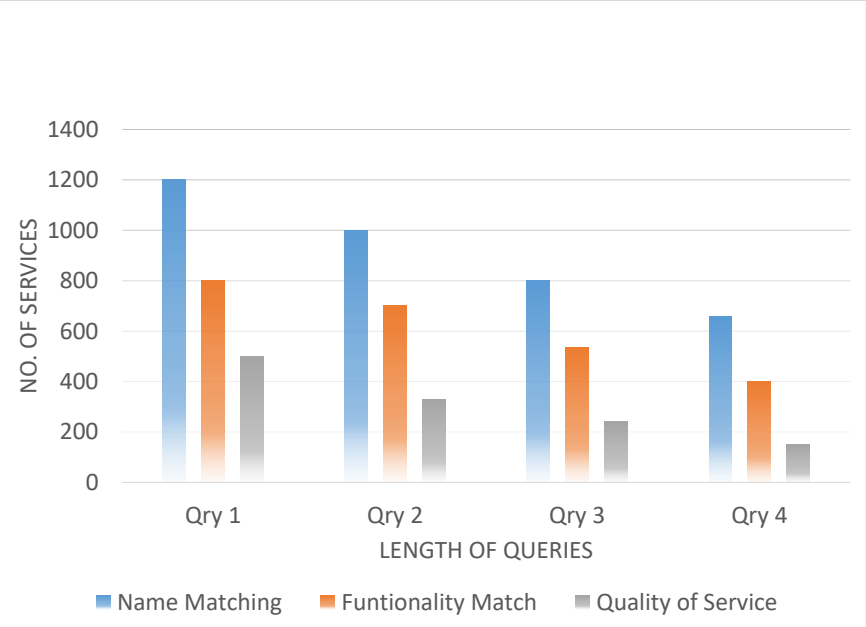


Figure 7. Number of matched services after each filtering.

5.4. Varying Location Range Versus Device Inclusion/Exclusion

Upon decreasing the radius of geographic search area, the number of candidate services decreases for service matching algorithm. This is based on the assumption that IoT devices and services are evenly distributed in a given geographic area. Thus when search is restricted in a particular area then limited number of IoT devices will be available for service matching. Conversely, as the area of geographic region of interest is increased, the number of IoT devices and services will increase and more appropriate services will be available for service matching.

Upon decreasing the radius of geographic region, the number of services decreases passing all the three matching criteria. The last filter for service matching is the most affected among all shown in Figure 8. The devices are spread randomly and by restricting the radius results in fewer devices and services.

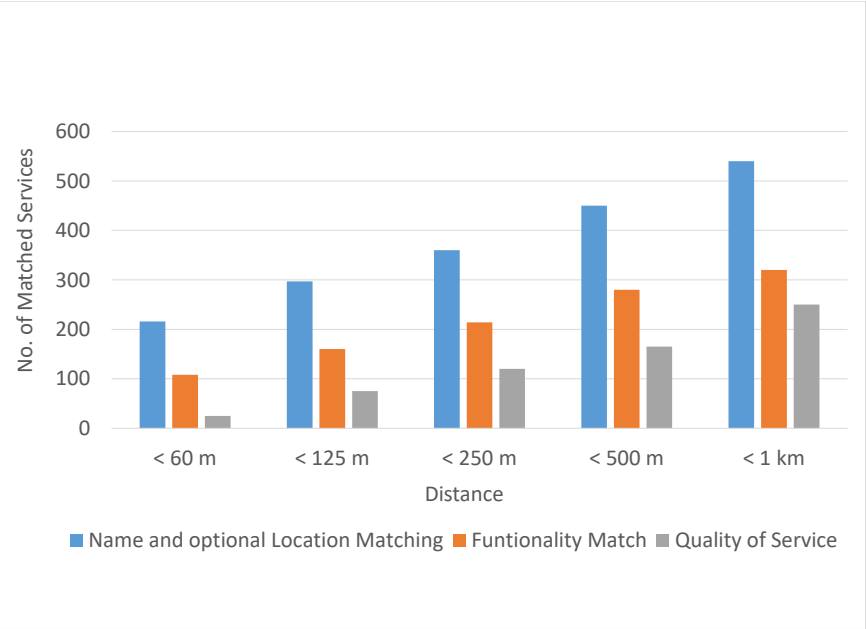


Figure 8. Device inclusion/exclusion based on distance.

5.5. Varying Threshold Versus Service Inclusion/Exclusion

Remaining energy is one of the most critical QoS parameter in desired services selection. Usually, we are interested in services matching a specified criteria having maximum remaining energy. Because, matching services having low remaining energy will quickly deplete and will result in services disruption. Moreover, selection of services with high remaining energy not only maximizes the overall network lifetime but also results in optimum utilization of available resources which is highly desirable.

The remaining energy of IoT devices is assumed to be uniformly distributed and, therefore, if we are interested in IoT devices or services with more residual energy, then a limited number of IoT devices or services will fulfill this criterion. Thus, the service matching algorithm will produce a limited number of IoT devices or services because fewer number of IoT devices are found to be fully charged in a given time period. A threshold value is specified for all types of devices. For example, remaining energy is an important attribute for IoT devices. In order to know about the energy level of different devices, the attribute of remaining energy expressed in millijouls (mJ) is used. The profiles of devices can provide us the remaining energy level of all devices.

In Figure 9, it is obvious that as we increase the threshold value from 10 mJ to 499 mJ, the number of devices decreases. Varying the threshold of remaining energy level shows that there are fewer devices which are fully charged in a given time period.

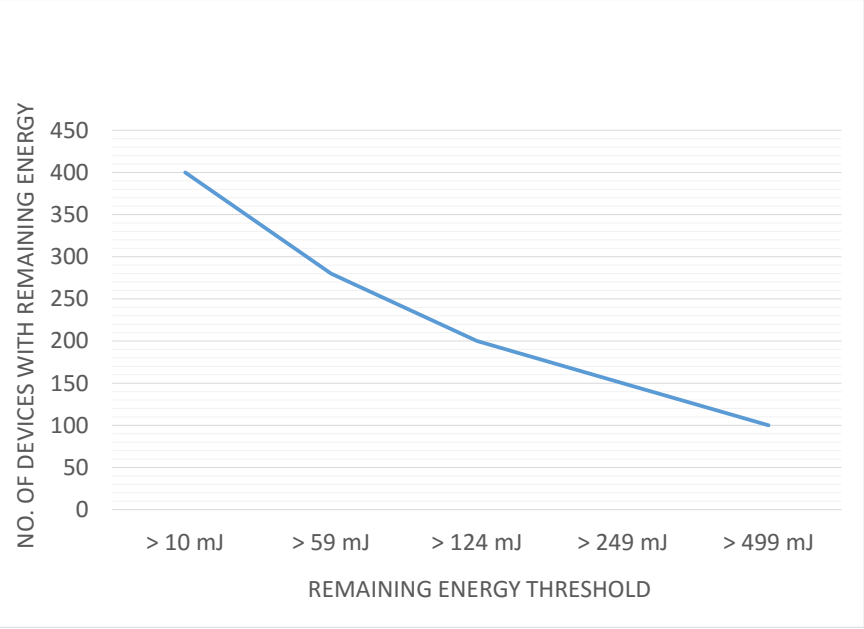


Figure 9. Devices remaining energy.

6. Conclusions

In this research, we contributed a virtualization based service matchmaking and discovery algorithm for IoT networks. The performance analysis of the proposed model is conducted in different situations. The matchmaking algorithm was implemented in Visual Studio 2015. A number of experiments were conducted to evaluate the performance concerning processing time for service matching, different query sizes and matched services at each filter, etc. This study has arrived at a few conclusions regarding the proposed service matchmaking algorithm utilizing 4000 IoT devices with 10,000 services. The increasing number of services in a given pool increased the average processing time for any particular service matching. Imposing restrictions on service matching also increased the overall processing time. Every stage of the proposed algorithm filtered out lesser services for final selection. As the devices were spread randomly, reducing the radius of geographic area popped out fewer devices and services by the proposed algorithm. At a given time, there was a decreasing trend

in IoT devices which were completely charged. In the future, the authors have the intention to work on the security issues in the given context.

Author Contributions: Conceptualization, Z.A.K. and I.U.; methodology, I.U. and A.A.; software, Z.A.K.; validation, M.I., M.F. and M.S.Q.; formal analysis, Z.A.K.; investigation, M.I. and A.A.; resources, M.F., M.S.Q.; data curation, M.I.; writing–original draft preparation, Z.A.K.; writing–review and editing, M.F., A.A. and M.S.Q.; visualization, I.U.; supervision, M.I. and I.U. All authors have read and agreed to the submitted version of the manuscript have contributed substantially to the work reported.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dua, R.; Raj, G. Quality Analysis for Web Services Recommendation Using Functional and Non-Functional Requirement. In Proceedings of the 3rd International Conference on Internet of Things and Connected Technologies (ICIoTCT), Jaipur, India, 26–27 March 2018; pp. 26–27.
2. Wilde, N.; Gonen, B.; El-Sheikh, E.; Zimmermann, A. Approaches to the evolution of SOA systems. In *Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures*; Springer: London, UK, 2016; pp. 5–21.
3. El-Sheikh, E.; Zimmermann, A.; Jain, L.C. *Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures*; Springer: London, UK, 2016.
4. Chen, F.; Lu, C.; Wu, H.; Li, M. A semantic similarity measure integrating multiple conceptual relationships for web service discovery. *Expert Syst. Appl.* **2017**, *67*, 19–31. [CrossRef]
5. Ullah, I.; Ahmad, S.; Mehmood, F.; Kim, D. Cloud Based IoT Network Virtualization for Supporting Dynamic Connectivity among Connected Devices. *Electronics* **2019**, *8*, 742. [CrossRef]
6. Shinde, G.; Olesen, H. A Survey on Service Discovery Mechanism. In *Intelligent Computing and Information and Communication*; Springer: London, UK, 2018; pp. 227–236.
7. Ullah, I.; Sohail Khan, M.; Kim, D. IoT Services and Virtual Objects Management in Hyperconnected Things Network. *Mob. Inf. Syst.* **2018**, *2018*. [CrossRef]
8. Osborne, G.; Weninger, T. Ozy: A general orchestration container. In Proceedings of the 2016 IEEE International Conference on Web Services (ICWS), San Francisco, CA, USA, 27 June–2 July 2016; pp. 609–616.
9. Cabrera, C.; White, G.; Palade, A.; Clarke, S. The right service at the right place: A service model for smart cities. In Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications (PerCom), Athens, Greece, 19–23 March 2018; pp. 1–10.
10. Xiao, H.; Zou, Y.; Tang, R.; Ng, J.; Nigul, L. An automatic approach for ontology-driven service composition. In Proceedings of the 2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), Taipei, Taiwan, 14–15 December 2009; pp. 1–8.
11. Liang, Y.; Zhou, X.; Yu, Z.; Wang, H.; Guo, B. A context-aware multimedia service scheduling framework in smart homes. *EURASIP J. Wirel. Commun. Netw.* **2012**, *2012*, 1–15. [CrossRef]
12. Chirila, S.; Lemnaru, C.; Dinsoreanu, M. Semantic-based IoT device discovery and recommendation mechanism. In Proceedings of the 2016 IEEE 12th International Conference on Intelligent Computer Communication and Processing (ICCP), Cluj-Napoca, Romania, 8–10 September 2016; pp. 111–116.
13. Pakari, S.; Kheirkhah, E.; Jalali, M. Web service discovery methods and techniques: A review. *Int. J. Comput. Sci. Eng. Inf. Technol.* **2014**, *4*. [CrossRef]
14. TalebiFard, P.; Leung, V.C. A data fusion approach to context-aware service delivery in heterogeneous network environments. *Procedia Comput. Sci.* **2011**, *5*, 312–319. [CrossRef]
15. Deng, J.; Chen, W.; Zhang, Z.; Xiang, T.; Zeng, R. A web service matching algorithm based on semantic similarity. *COMPEL-Int. J. Comput. Math. Electr. Electron. Eng.* **2013**, *32*.
16. Gmati, F.E.; Ayadi, N.Y.; Bahri, A.; Chakhar, S.; Ishizaka, A. Customizable Web services matching and ranking tool: Implementation and evaluation. In *International Conference on Web Information Systems and Technologies*; Springer: London, UK, 2016; pp. 15–36.

17. Nan, C.; Tila, F.; Kim, D.H.; Khan, M.S.; Kang, H.Y.; Park, D.H.; Lee, J.H. A Study of Integration Architecture Based on Service Provider in Heterogeneous Sensor Networks. In Proceedings of the 2015 8th International Conference on Disaster Recovery and Business Continuity (DRBC), Jeju Island, Korea, 25–28 November 2015; pp. 1–3.
18. Khan, M.S.; Kim, D. Service-oriented process modelling for device control in future networks. *Int. Arab J. Inf. Technol.* **2017**, *14*, 691–697.
19. Jia, B.; Li, W.; Zhou, T. A centralized service discovery algorithm via multi-stage semantic service matching in Internet of Things. In Proceedings of the 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, China, 22–23 July 2017; Volume 1, pp. 422–427.
20. Calcina-Ccori, P.C.; De Biase, L.C.C.; Fedrecheski, G.; da Silva, F.S.C.; Zuffo, M.K. Enabling semantic discovery in the swarm. *IEEE Trans. Consum. Electron.* **2018**, *65*, 57–63. [CrossRef]
21. Ahmed, A.H.; Omar, N.M.; Ibrahim, H.M. Secured Service Discovery Technique in IoT. *JCM* **2019**, *14*, 40–46. [CrossRef]
22. VisualStudio, October 2019. 2016. Available online: https://visualstudio.microsoft.com/vs/older-downloads/ (accessed on 21 October 2019).



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).