EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

Publication and Discovery of Things in the Internet of Things

Naseem Ibrahim[1,*]

[1]Penn State Behrend, Erie PA, USA

Abstract

A large number of IoT platforms have been introduced in the last few years. The major issue with available platforms is that they view Things as private property. But this view is short sighted. A Thing should be viewed as a service or a functionality provider. The Thing owner should be able to publish Thing information and make it public. On the other hand, Thing requesters should be able to discover available Things. Our research is concerned with introducing an architecture that supports the publication and discovery of Things. This architecture will enable Thing owners to publish Thing specification and Thing requesters to find Things that best matches their needs. The main contribution of this paper is an SOA Context-aware Architecture for the publication and discovery of Things. This paper formally defines an architecture for Things and queries. Also, this paper presents two XML languages that supports the publication and query of Things.

Received on 11 January 2018; accepted on 04 March 2018; published on 20 March 2018

Keywords: Internet of Things, Service-oriented Architecture, Context, Publication, Discovery

doi:10.4108/eai.20-12-2018.156082

1. Introduction

The Internet of Things is turning into an explosive topic in today's technology world. Smart Things are becoming more prevalent in both business and personal use. From cell phones, smart homes, smart sensors and more [5], nearly every object can be made intelligent and relay information and data to a user at a moment's notice. Billions [14] of new smart Things are expected to take hold in the next few years. Each Thing can provide a functionality or a service to a user or client. You don't have to own a Thing to use its functionality or service. A user should be able to search for Things that provide a required functionality. But with the massive growth of smart Things, it will become difficult for users to find these smart Things and more importantly find one or more that best meets the user's requirements.

For example, a person driving a vehicle downtown looking for a parking spot. How will the user find an available parking spot without having to drive around? How will the user find the most convenient parking spot? A spot that is near, available, within the driver's budget, and with a size that aligns with the type of vehicle they are driving. Smart Parking Lots and Smart Parking Spaces can be used to determine if a parking spot is open. This information paired with the location, time, fee, and spot size can be used to match a driver with a spot that meets their requirements and then directs them to the spot. This can not be accomplished using the current architecture of the Internet of Things.

An architectural model in which *service* is a first class element for providing functionality is called *Service-oriented Architecture (SOA)* [2]. SOA can be thought of as a group of individual services that blend and interact with other services seamlessly, to accomplish a task. Context awareness [7] is the ability to detect and respond to changes in context. Context [1] is the conditions that are used to describe a product, or a service. This paper proposes representing and publishing the functionalities provided by a Thing as a service. The service should be rich enough to represent the changing properties of Things, hence the paper added context-awareness to the process of publishing and discovering the services provided by the Things.

The rest of this paper is structured as follows. Section 1.1, introduces the problem and the proposed solution. Section 2, gives a brief introduction to the Internet of Things, SOA, and context. Section 3, introduces the novel architecture for the provision of services in the Internet of Things. Section 4,

*Corresponding author. Email: nii1@psu.edu

introduces the structure of the context-aware service that is used to publish the functionality provided by Things. Section 5, introduces the structure of the context-aware query that is used to query Things. Section 6, introduces the Thing Specification Language. Section 7, introduces the Thing Query Language. Section 8, provides a brief introduction of related work. Finally, Section 9 provides some concluding remarks and a brief discussion of future work.

1.1. Problem and Contributions

In the Internet of Things, Things provide services or functionality. A user does not need to own a Thing to get a functionality. A user should be able to search for all available Things. But for this process to work, the owner of the Thing should make the Thing information public and the user of Thing should be able to find the Thing information. With the current architecture of the Internet of Things this is not possible. *This paper investigates achieving the publication and discovery of Things using SOA*. Combining SOA with the Internet of Things will allow users to request and receive functionalities and services from smart objects in a seamless manner.

However, as the number of Things and users increase, it will become difficult to pair a Thing with a user. It will become even more difficult to find a smart object that will meet the user's requirements and context. Because of this, the process will become time consuming and less efficient. In order to find the Thing that will provide the user with the most relevant service, the matching process between the user's requirements and available Things must consider context. Some examples of context with regards to users are location, time, weather, mood, age, etc. With context awareness, these smart Things will publish their context, and Things will be selected using the context of the Thing and user.

This paper aims to solve the problem of scalable *publishing and discovery* of smart Things using SOA. It solves the problem of scalable *matching* of available Things with the use of context-awareness. The result will be a Context-aware Service-oriented Architecture that will allow users to be paired with Things based on their context and requirements. This will lead to more accurate data and information coming from smart objects and services. This will decrease the time the user spends searching for the services and Things they need to use.

The main contribution of this paper is a context-aware service-oriented architecture that will support the publication, discovery and matching of Things in the Internet of Things. This architecture will enable Things owners to publish the information of their Things in a rich definition using context. It will also enable users or consumers to discover these things. A matching that will consider the context of Things and consumers will ensure the best match between the user's requirements and available Things.

To support the publication process, we introduce a new architecture for the specification of Things. This architecture is formalized to support the formal verification of the claims made in the Thing description. To support portability and implementation, we introduce an XML based language for the specification of Things called Thing Specification Language(TSL).

To support the discovery process, we introduce a new architecture for the specification of requesters requirements. This architecture is formalized to support the formal verification of the requirements specified in the requester query. To support portability and implementation, we introduce an XML based language for the specification of Queries called Thing Query Language (TQL).

2. Background

This section gives a brief introduction to the Internet of Things, Service-oriented Architecture and Context. It will lay the basis for the rest of the paper.

2.1. Internet of Things

The Internet of Things (IoT) [5] is the connection of physical objects that collect and transfer data from one object to another. These objects are embedded with software, and sensors. These objects can be virtually anything. There is machine to machine communication, and communication between sensors and machines. These sensors gather data, and then communicate that to another object that can then analyze the data and make decisions. Most of today's IoT applications [14] are cloud-based. These cloud applications [2] combine the characteristics and functionality of a desktop, and web application by offering portability, and usability. These applications [14] are key in receiving the data, interpreting it, and transmitting it to create useful information and decisions. The idea of the Internet of Things is a massive fundamental shift. Numerous possibilities of new products and services are introduced by these intelligent Things.

The Internet of Things is expected to change the day to day operations of virtually everything. IoT can be divided into 5 broad categories [5]: smart wearable, smart home, smart city, smart environment, and smart enterprise.

The main issue with current IoT architectures is that they are looking at Things as private property. An organization or an individual owns a Things, they use the functionalities provided by the Thing to achieve a task or to make a decision. Basically, only the owner of the Thing can make any use of the Thing. We believe this is counter productive, Things can be service providers to anyone. Any client should be able to find a Thing that provides a service that the client needs. This service can be an actual physical tasks, information needed for a decision or a cyber task. Thing owners should be able to publish the services provided by their Things, and clients should be able to find Things that provide the required services. But current architectures for the implementation of IoT does not support this philosophy. Hence, this paper investigates the use of service-oriented architecture.

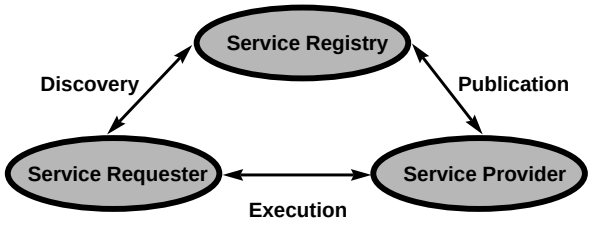

Figure 1. Traditional SOA Architecture

2.2. SOA

Service-oriented Computing (SOC) [4] is a computing paradigm that uses *service* as the fundamental element for application development processes. An architectural model of SOC in which *service* is a first class element is called *Service-oriented Architecture* [2]. Service-oriented architecture (SOA) [8] is the collection, communication, and sharing of services. It can involve sharing data, or coordinating activity when connected. A service is a task well defined, that does not depend on the state of other services. SOA consists of three main modules, the service provider, the service requester and the service registry. The service provider publishes a service definition in the service registry. The service requester searches the service registry and selects from the published services. After selecting a service, the service requester interacts with the service provider by sending requests and receiving responses.

Some benefits [4] of SOA include: platform independence, code reuse, easier testability, easier developer focus, and parallel development. Amazon [10] has transformed into an SOA company, they also have a universal service registry. Amazon now thinks about everything in a service first fashion. Accessibility is the most important thing in the computing world. Scalability with contracts is also important in SOA. Amazon web pages [6] communicate with around 150 services. Smaller, and simpler tools are more easily scalable.

Figure 1 illustrates the three main activities in SOA are *service publication*, *service discovery* and *service provision*. In the literature, these terms are used in the following sense. *Service publication* refers to defining the service contract by service providers and publishing them through available service registries. *Service discovery* refers to the process of finding services that have been previously published and that meet the requirements of a service requester [13]. Typically, service discovery includes *service query, service matching,* and *service ranking*. Service requesters define their requirements as service queries. Service matching refers to the process of matching the service requester requirements, as defined in the service query, with the published services. Service ranking is the process of ordering the matched services according to the degree they meet the requester requirements. The ranking will enable the service requester to select the most relevant service from the list of candidate services. *Service provision* refers to the process of executing a selected service. The execution may include some form of an interaction between the service requester and service provider. This paper investigates the use of SOA for the publication, discovery, and provision of the services provided by Things in the Internet of Things.

Devices need to communicate information and data to each other. The seamless integration of applications and different services are crucial. SOA can also be useful in combining business and IT. You can take objects and applications, and use services to add more functionality to an application. You can send data [11] to a service and have that service analyze information, without having to develop similar software and have that infrastructure. IoT devices themselves [17], can be considered services to enterprises, society, and industries. You can use Facebook service as a platform to have users log information. You can use services to process payment information. This allows modularity, the developer does not have to worry about these services. SOA deals with IoT in a business way. SOA can allow IoT and its applications to have far more features by using seamless services.

2.3. Context

Context has been defined [1] as the information used to characterize the situation of an entity. This entity can be a person, a place, or an object. Context is an important concept to consider when designing and implementing software. There are multiple different contexts to consider. One of the most important is the size of the system which will dictate multiple choices in the design process. Having stable architecture, using a correct business model, assigning good team distribution, keeping in mind the age of the system, safety, and governance are other examples of needing context in projects.

With the emergence of IoT, context [15] has become even more important. Some questions to ask when determining context is who will be using it, why, when, where, and what. It also consists of more specific aspects such as [19] location, time, temperature, even users emotions and preferences; these can be classified as scalar, vector, or abstract values. Selecting certain factors such as platform, will eliminate a lot of context of use issues; however, when considering IoT, most cases will not be platform dependent. An example [15] of context of use for an IoT design would be considering a heart rate device. Some context information that developers would want to ask for are: to make it portable or wearable, the display of the monitor, what inputs it should take, what type of consumer would use this, and what is the best way the information can be shared. Another example for the context of use is when a driver is driving in a car using navigation or other features. Developers would need to consider the interface, too simple may not provide enough functionality, but too complex and it becomes dangerous for the driver.

Smart devices being context aware is another interesting concept. Cell phones being completely context aware would

require sensors, and RFID tags. This would allow phones to be aware of locations and other information [19], which will limit the amount of interaction between the user and device. Context can also be used in terms of service contracts which is responsible for the fulfillment of agreed upon conditions.

3. Publication and Discovery Architecture

This section reviews our novel SOA based architecture that was introduced in [9]. This architecture supports the publication and discovery of Things in the Internet of Things. The main goal of this architecture is support the publication and discovery of trustworthy context-dependent Things. To achieve this goal the architecture should consider the following criteria:

- *Context:* Things are context-aware in nature. The consideration of context information in publication and discovery is sessional to find the best match between the requester requirements and the available Things.
- *Dynamic context*: The context information of requesters and Things are not static, they can dynamically change. The architecture should support dynamic change to best match clients and Things.
- *Scalability:* As the number of Things increase drastically, it will be important that the architecture can adapt and scale with this increase.
- *Discovery:* In order for a client to find Things, the architecture must support the discovery of Things. Discovering Things can be achieved by discovering the services provided by the Things.
- *Search:* After discovering the services provided by Things, the client should be able to search for the Things that best matches his/her needs.
- *Ranking:* The search process will be time consuming with the increased number of Things being published. To really make this architecture useful, the architecture should rank the result of the search according to the context of the Thing and client. This ranking will decrease the amount of time the user spends searching for the service they desire. This ranking will also provide the user with the most relevant Things to them based on their context and needs.
- *Formality:* The architecture should be formal. Formalism will enable the validation and verification of the functions provided by the architecture.

Figure 2 illustrates our newly introduced SOA based architecture for the publication, and discovery, of Things in the Internet of Things. This architecture satisfies the seven criteria discussed above. Below is a brief review of the elements of the architecture.

Registry: The registry is the entity that will be responsible for the publication, discovery, matching and ranking of things.

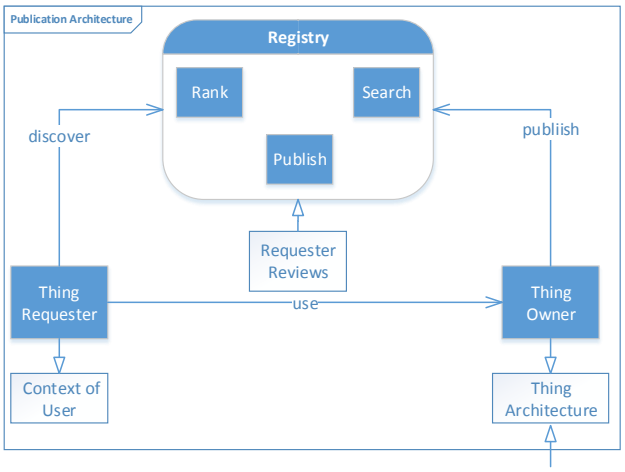

Figure 2. SOA-based IoT Architecture

It will allow the Thing owner to publish his Things, and users to discover these Things. The Registry consists of four main elements:

- *Search:* This element is responsible for the discovery of Things. It will enable clients to find Things that provide the services they are requesting. The discovery will be based on the client context, and the context of published Things.
- *Rank:* This element is responsible for ranking the Things that meets the client's requirements. It is very common that the registry will find multiple Things that meets the client's requirements. It is also common that none of these Things provide an exact match to the client's requirements. Hence, a ranking is necessary. The ranking element will run ranking algorithms that will rank Things according to the client's context and priorities.
- *Publish:* This element will be responsible for storing the information provided by the owners of Things. The information provided by the providers will be structured according to the Thing structure discussed in the next section.
- *Requestor Reviews:* To ensure the correctness of the information contained in the registry. The registry will also allow Thing users to review Things after using them. The reviews completed by users will be managed by this element. The review can include, how the service performed, the correctness of the description, and any experiences the client have with the Thing owner.

Requester: This entity represents the client that is trying to discover a Thing that provides a required functionality or service. The requester will directly interact with the registry.

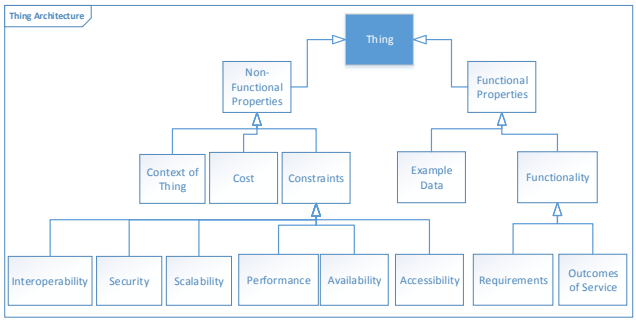

Figure 3. IoT Thing Architecture

After discovering a Thing, the requester can use the Thing by directly interacting with it. The requester provides his context information to the registry to get the best match. Hence, the requester interacts with the following entity.

- *Context of User:* This entity is responsible for collecting and structuring the context of the user. This entity is dynamic. It monitors the user context and update the context information accordingly. The context information provided by this entity will be used by the client when communicating with the registry.

Owner: This entity represents the owner of the Thing. The owner of the Thing will publish the Thing information in the registry to be available for discovery. The Thing owner will also enable the communication between clients and their Things. The information published by the Thing owner will be structured according to the Thing Architecture.

- *Thing Architecture:* This entity will define the structure that the Thing owner should use to describe his Things. This is discussed in details in the next section.

4. Thing Architecture

The previous section discussed the architecture for the publication, discovery, and provision of Things. In this architecture, the Thing owner will publish the Thing information in the registry. This section introduces the structure of the Thing information that will be published. This section introduces the informal and formal description of a Thing.

4.1. Informal description of Things

To ensure consistency and correctness, the information of the Thing will be structured according to the structure discussed in this section. Figure 3 illustrates the architecture for a Thing that can be published and discovered using our SOA based architecture. A Thing consists of two main elements, non-functional and functional properties. Following is a discussion of the main elements of the Thing architecture.

Functional Properties: This element define the behavior and functionality provided by a Thing. It consists of the following entities.

- *Functionality:* This element will include the following entities.
  - *Outcomes of Service:* This entity states the postconditions that are guaranteed by the Thing to be true after execution. It can also include the technologies used by the Thing.
  - *Requirements:* This entity describes the preconditions that should be satisfied by the Thing user before execution.
- *Example Data:* This entity shows what the output will look like, and how it will be formatted.

Non-Functional Properties: This element specifics the non-functional properties that will constrain the functionality provided by the Thing. This element is divided into the following entities.

- *Context of Thing:* This entity will describe all of the Thing's context, the context used will be dependent upon the Thing. This will be both static and dynamic context. The point of having context will be to aid in pairing a requestor with the proper service based on their context.
- *Cost:* This entity will state the cost of requesting a functionality from a Thing. This can be either for a one-time use, or a subscription based service.
- *Constraints:* This entity will contain the non-functional guarantees that the Thing can provide. It includes the following.
  - Interoperability: This will define if the Thing has the ability to work with other systems or products (the service will use or be used by other services). It can also include information related to platform support.
  - Security: This will describe the security mechanism used to secure data, and communication.
  - Scalability: This discuss the Thing ability to scale to a large number of requests. It can contain the maximum number of requests it can handle per unit of time.
  - Performance: This will display the performance information such as: How long the process takes? and How accurate is the data being received?
  - Availability: This will describe the maximum amount of down time that is guaranteed by the Thing provider. It can also include the operating hours.

– Accessibility: This can include information about user location constraints, type of users allowed and Thing location.

4.2. Formal description of Things

A *Thing* is formally defined using a model-based specification notation. The context information encapsulated in the non-functional part of the Thing is written in the notation introduced by Wan [18]. The formal definition will enable the verification of the claims made in the Thing description. Below we give the formal representation of the Thing elements.

Let $\mathbb{C}$ denote the set of all such logical expressions. $X \in \mathbb{C}$ is a constraint. The following notation is used in our definition:

- $\mathbb{T}$ denotes the set of all data types, including abstract data types.
- $Dt \in \mathbb{T}$ means Dt is a datatype.
- $v : Dt$ denotes that v is either constant or variable of type Dt.
- X_v is a constraint on v. If v is a constant then X_v is true.
- V_q denotes the set of values of data type q.
- $x :: \Delta$ denotes a logical expression $x \in \mathbb{C}$ defined over the set of parameters Δ. A parameter is a 3-tuple, defining a data type, a variable of that type, and a constraint on the values assumed by the variable. We denote the set of data parameters as $\Lambda = \{\lambda = (Dt, v, X_v) | Dt \in \mathbb{T}, v : Dt, X_v \in \mathbb{C}\}$.

Functional Properties. The functional properties section encapsulate the *Functionality* and *Example data*. A Thing can provide a single functionality. This functionality is defined to include the function *Requirements* and *Outcome of Service*.

Definition 1. A Thing functional properties is a 2-tuple $fp = \langle f, r \rangle$, where f is the functionality, and r is the function example data which represents the result format. The example data is defined as $r = \langle m, q \rangle$, where $m : string$ is the result identification name and $q = \{x | x \in \Lambda\}$ is the set of parameters. A functionality is a 2-tuple $f = \langle re, ot \rangle$ where requirements re and outcomes ot are data constraints. That is, $re :: z, z \subseteq \Lambda$ and $ot :: z, z \subseteq \Lambda$

Non-functional properties. The non-functional properties section encapsulates the following:

Context: Context information and Context rules are formally specified as part of the context of a Thing. These two parts provide context-awareness ability to Things. Context information is formally specified, as defined in [18], using *dimensions* and *tags* along the dimensions. In our research, we have been using the five dimensions *where*, *when*. *what*, *who*, and *why*. Assume that the service provider has invented a finite set $DIM = \{X_1, X_2, \ldots, X_n\}$ of dimensions, and associated with each dimension X_i a type τ_i. Following the formal aspects of context developed by Wan [18], we define a context c as an aggregation of ordered pairs (X_j, v_j), where $X_j \in DIM$, and $v_j \in \tau_j$. A Context rule is a situation which might be true in some contexts and false in some others. For example, the situation $warm = temp > 30 \ \wedge \ humid > 75$, is true only in contexts where the temperature is greater than 30 degrees, and the humidity is greater than 75.

Definition 2. A context is formalized as a 2-tuple $\beta = \langle r, c \rangle$, where $r \in \mathbb{C}$, built over the contextual information c. Context information is formalized using the notation in [18]: Let $\tau : DIM \rightarrow I$, where $DIM = \{X_1, X_2, ..., X_n\}$ is a finite set of dimensions and $I = \{a_1, a_2, ..., a_n\}$ is a set of types. The function τ associates a dimension to a type. Let $\tau(X_i) = a_i$, $a_i \in I$. We write c as an aggregation of ordered pairs (X_j, v_j), where $X_j \in DIM$, and $v_j \in \tau(X_j)$.

Cost: The elements specifies the cost for requesting a functionality from a Thing.

Definition 3. The service cost p is defined as a 3-tuple $p = \langle a, cu, un \rangle$, where $a : \mathbb{N}$ is the price amount defined as a natural number, $cu : cType$ is currency tied to a currency type $cType$, and $un : uType$ is the unit for which pricing is valid. As an example, $p = (100, \$, hour)$ denotes the pricing of 100$/hour.

Constraints: This element represents the trustworthiness and quality properties that Thing providers guarantees to be met before, during and after the provision of the functionality of the Thing.

Definition 4. A constrains is a rule, expressed as a logical expression in $\mathbb{C}$. A rule may imply another, however no two rules can conflict. We write $con = \{y | y \in \mathbb{C}\}$ to represent the set of Constraints.

Putting these definitions together we arrive at a formal definition for the Thing non-functional properties.

Definition 5. Non-Functional properties is a 3-tuple $nf = \langle \beta, p, con \rangle$, where β is the context of the Thing, c is the cost of the Thing, and con is the set of constraints.

Finally, the formal definition of a Thing is:

Definition 6. A Thing is a 2-tuple $Th = \langle fp, nf \rangle$, where fp is the functional properties of the Thing, and nf is the nonfunctional properties.

5. Thing Query

The previous section briefly introduced the architecture for the publication and discovery of Things. In this architecture, the Thing owner will publish the Thing description in the registry. The Thing requester will then query the Registry looking for Things that will satisfy a required functionality. This section introduced the structure of the Thing query informally and formally.

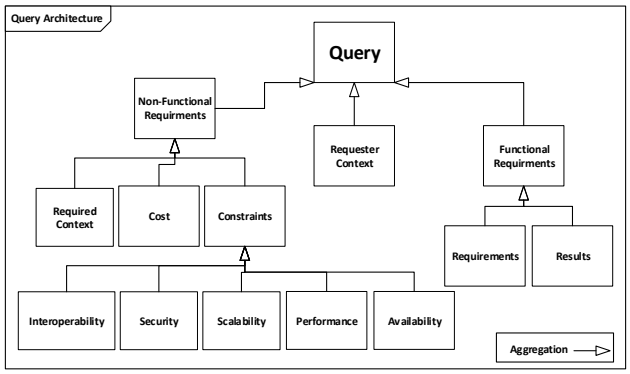

Figure 4. Query Architecture

5.1. Informal description of Query

To ensure consistency and correctness, a Thing query will be structured according to the structure discussed in this section. This subsection will introduce an informal description of the Thing query structure. Figure 4 illustrates the architecture for a Thing query. A Thing query consists of three main elements, non-functional requirements, functional requirement, and requester context. Following is a discussion of the main elements of the Thing query.

Functional Requirement: This element specifies the requesters required functionality. The functional requirement will be used by the registry to match the requester required functionality with available Things. A functional requirements is defined in terms of requirements and results.

- *Requirements:* This entity describes the preconditions that the Thing requester guarantees to satisfy before requesting a Thing functionality.

- *Results:* This entity describes the postconditions that the requester is requiring to be satisfied.

Non-Functional Requirements: This element specifics the non-functional properties that the Thing requester is requiring. It is important to emphasize that for each of these requirements the Thing requester can specify a priority. A priority is a Natural number between 1 and 5, where 1 indicates low priority and 5 indicates a high priority. The priority will enable the Registry to rank the matched Things according to the matched functionality and non-functional properties. This non-functional requirements elements is divided into the following entities.

- *Required Context:* This entity will describe any conditions or rules the requester will have with respect to the Thing context. This will be both static and dynamic context. This will also enable the matching of the required context with the Thing published context.

- *Cost:* This entity will state the maximum accepted cost of a Thing functionality. This can be either a one-time use cost, or a subscription cost.

- *Constraints:* This entity will contain the non-functional guarantees that the requester is requesting. It includes the following.

  - Interoperability: This will define the required platforms or environments.
  - Security: This will describe the required security mechanisms to secure data, and communication.
  - Scalability: This discuss the required Thing ability to scale to a large number of requests. It can contain the maximum number of requests it can handle per unit of time.
  - Performance: This will display the required performance information.
  - Availability: This will describe the maximum acceptable down time that is required by the Thing requester. It can also include the required operating hours.

Requester Context: This entity will describe the requester's context. This will be both static and dynamic context. The point of having context will be to aid in pairing a requestor with the proper Thing based on their context information.

5.2. Formal description of Query

A *Query* is formally defined using a model-based specification notation. The context information is written in the notation introduced by Wan [18]. Below we give the formal representation of the Thing Query.

Let $\mathbb{C}$ denote the set of all such logical expressions. $X \in \mathbb{C}$ is a constraint. The following notation is used in our definition:

- $\mathbb{T}$ denotes the set of all data types, including abstract data types.

- $Dt \in \mathbb{T}$ means Dt is a datatype.

- $v : Dt$ denotes that v is either constant or variable of type Dt.

- X_v is a constraint on v. If v is a constant then X_v is true.

- V_q denotes the set of values of data type q.

- $x :: \Delta$ denotes a logical expression $x \in \mathbb{C}$ defined over the set of parameters Δ. A parameter is a 3-tuple, defining a data type, a variable of that type, and a constraint on the values assumed by the variable. We denote the set of data parameters as $\Lambda = \{\lambda = (Dt, v, X_v) | Dt \in \mathbb{T}, v : Dt, X_v \in \mathbb{C}\}$.

Functional Requirement. : The functional requirement section encapsulate the function *Requirements* and *Results*. A request includes a single functionality.

Definition 7. A functional requirement is a 2-tuple $\hat{f} = \langle \hat{re}, \hat{ot} \rangle$ where requirements $\hat{re}$ and results $\hat{ot}$ are data constraints. That is, $\hat{re} :: z, z \subseteq \Lambda$ and $\hat{ot} :: z, z \subseteq \Lambda$

Requester Context:. This information is formally specified, as defined in [18], using *dimensions* and *tags* along the dimensions. In our research, we have been using the five dimensions *where*, *when*. *what*, *who*, and *why*. Assume that the service provider has invented a finite set $DIM = \{X_1, X_2, \ldots, X_n\}$ of dimensions, and associated with each dimension X_i a type τ_i. Following the formal aspects of context developed by Wan [18], we define a context c as an aggregation of ordered pairs (X_j, v_j), where $X_j \in DIM$, and $v_j \in \tau_j$.

Definition 8. Context information $\hat{c}$ is formalized using the notation in [18]: Let $\tau : DIM \rightarrow I$, where $DIM = \{X_1, X_2,...,X_n\}$ is a finite set of dimensions and $I = \{a_1, a_2, ..., a_n\}$ is a set of types. The function τ associates a dimension to a type. Let $\tau(X_i) = a_i$, $a_i \in I$. We write $\hat{c}$ as an aggregation of ordered pairs (X_j, v_j), where $X_j \in DIM$, and $v_j \in \tau(X_j)$.

Nonfunctional Requirements. : The non-functional requirements section encapsulates the following:

Requested Context: A requested context is the set of context rules, where a context rule is a situation which might be true in some contexts and false in some others. For example, the situation $warm = Temp > 25 \wedge humid > 50$, is true only in contexts where the temperature is greater than 25 degrees, and the humidity is greater than 50.

Definition 9. A requested context is formalized as $\hat{r}$ where $\hat{r} = \{x | x \in \mathbb{C}\}$.

Cost: The elements specifies the maximum acceptable cost for requesting a functionality from a Thing.

Definition 10. The cost $\hat{p}$ is defined as a 3-tuple $\hat{p} = \langle a, cu, un \rangle$, where $a : \mathbb{N}$ is the price amount defined as a natural number, $cu : cType$ is currency tied to a currency type $cType$, and $un : uType$ is the unit for which pricing is valid. As an example, $\hat{p} = (25, \$, hour)$ denotes the pricing of 25$/hour.

Constraints: This element represents the trustworthiness and quality requirements. Each requirement can be mapped to a priority.

Definition 11. A constrains is a rule, expressed as a logical expression in $\mathbb{C}$. A rule may imply another, however no two rules can conflict. We write $\hat{con} = \{y | y \in \mathbb{C}\}$ to represent the set of constraints.

Putting these definitions together we arrive at a formal definition for the non-functional requirements.

Definition 12. Non-Functional requirements is a 4-tuple $\hat{nf} = \langle \hat{r}, \hat{p}, \hat{con}, \Xi \rangle$, where $\hat{r}$ is the required context, $\hat{p}$ is the required cost of the Thing, $\hat{con}$ is the set of constraints, and $\Xi : (x \in \{1, 2, 3, 4, 5\}) \rightarrow (y \in \{\hat{c}, \hat{r}, \hat{p}, \hat{con}\})$ is a function that assign priorities to the elements of the query.

Finally, the formal definition of a query is:

Definition 13. A query is a 3-tuple $q = \langle \hat{f}, \hat{nf}, \hat{c} \rangle$, where $\hat{f}$ is the functional requirement, $\hat{nf}$ is the non-functional requirements, and $\hat{c}$ is the requester context.

6. Thing Specification Language

A Thing provider specifies the Thing specification using the architecture we introduced in the previous section. The Thing description has been formally defined to support formal verification. To support portability, in this section we introduce an XML based language called Thing Specification Language (TSL). Following is a discussion of the structure of TSL.

TSL represents a one-to-one mapping to the Thing structure presented in Figure 3. The two main elements of a Thing are functional properties and non-functional properties. Below is the XML schema for the main elements of a Thing.

```
<xsd:element name="Thing">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="thingName" type="xsd:string"/>
      <xsd:element name="functionalProperties">
        <xsd:complexType>
          ......
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="nonFunctionalProperties">
        <xsd:complexType>
          ......
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Below is a detailed discussion of the XML schema for each of these components.

6.1. Functional Properties

The functional properties definition includes the *Functionality*, *Example Data*. The *Functionality* part defines the function *requirements*, and *outcomes*. Each *Example Data* is defined with an *ID* and *parameter* list. Below is the XML schema for presenting functionality.

```
<xsd:element name="functionalProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="exampleData">
        <xsd:sequence>
          <xsd:element name="ID" type="xs:string"/>
          <xsd:complexType name="Parameter">
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string"/>
              <xsd:element name="DataType" type="
                  xsd:string"/>
              <xsd:element name="Value" minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:sequence>
      </xsd:element>
      <xsd:element name="functionality">
```

```
        <xsd:complexType>
          <xsd:sequence>
            <xsd:complexType name="requirements" minOccurs
                ="0" maxOccurs="unbounded">
              <xsd:sequence>
                <xsd:element name="condition" type="
                    xsd:string"/>
              </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="outcomes" minOccurs="0"
                maxOccurs="unbounded">
              <xsd:sequence>
                <xsd:element name="condition" type="
                    xsd:string"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

6.2. Non-functional Properties

The nonfunctional properties associated with the Thing are listed in this section. It includes, context, cost and constraints. The XML schema for the structure of the non-functional properties is shown below.

```
<xsd:element name="nonFunctionalProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="cost">
        .......
      </xsd:element>
      <xsd:element name="contextOfThing">
        .......
      </xsd:element>
      <xsd:element name="constraints">
        .......
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Cost. Cost information, which can itself be a complex property expressing different prices for different amount of buying, is a non-functional property. Below is the XML schema for specify the non-functional property cost.

```
<xsd:element name="cost">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:double"/>
      <xsd:element name="currency" type="xsd:string"/>
      <xsd:element name="unit" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Context of Thing. The context part is divided into *context info* and *context rules*. The contextual information of the Thing provider is specified in the *context info* section. The situation or context rule that should be true for Thing delivery is specified in *context rules* section. Below is the XML schema for defining context.

```
<xsd:element name="contextOfThing">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:complexType name="Context">
        <xsd:sequence>
          <xsd:element name="Location" type="String"
              minOccurs="0" maxOccurs="unbounded"/>
          <xsd:element name="Time" type="xsd:time"
              minOccurs="0"/>
          <xsd:element name="Date" type="xsd:date"
              minOccurs="0"/>
          <xsd:element name="WhoProvider" type="xsd:string
              " minOccurs="0"/>
          <xsd:element name="WhoRequester" type="
              xsd:string" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="ContextRules">
        <xsd:sequence>
          <xsd:element name="ContextRule" type="Context"
              minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Constraints. This section lists the guarantees that a Thing owner can provide to the requesters of the Thing functionalities. It includes interoperability, security, scalability, performance, availability and accessibility constraints. Below is the XML schema for specifying constraints.

```
<xsd:element name="constraints">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Interoperability" type="
          xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Security" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Scalability" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Performance" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Availability" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Accessibility" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

7. Thing Query Language

A Thing requester specifies the Thing requirements according to the architecture we introduced in the previous section. The Thing query has been formally defined to support formal verification. To support portability, in this section we introduce an XML based language called Thing Query Language (TQL). Following is discussion of the structure of TQL.

TQL represents a one-to-one mapping to the query structure presented in Figure 4. The three main elements of a query are functional requirements, non-functional requirements, and requester context. Below is the XML schema for the main elements of a query.

```
<xsd:element name="Query">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="functionalRequirements">
        <xsd:complexType>
          ......
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="nonFunctionalRequirements">
        <xsd:complexType>
```

```
        ......
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="requesterContext">
      <xsd:complexType>
        ......
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Below is a detailed discussion of the XML schema for each of these components.

7.1. Functional Requirements

The required function definition includes the *requirements*, and *results*. Requirements and results are logical conditions that can evaluate to either true or false.

```
<xsd:element name="functionalProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:complexType name="requirements">
        <xsd:sequence>
          <xsd:element name="condition" type="xsd:string"
              minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="results">
        <xsd:sequence>
          <xsd:element name="condition" type="xsd:string"
              minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

7.2. Requester Context

The contextual information of the Thing requester is listed in this section. It is defined according to the five dimensions discussed earlier. Below is the XML schema for defining the requester context.

```
<xsd:element name="requesterContext">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:complexType name="Context">
        <xsd:sequence>
          <xsd:element name="Location" type="xsd:string"
              minOccurs="0"/>
          <xsd:element name="Time" type="xsd:time"
              minOccurs="0"/>
          <xsd:element name="Date" type="xsd:date"
              minOccurs="0"/>
          <xsd:element name="WhoProvider" type="xsd:string
              " minOccurs="0"/>
          <xsd:element name="WhoRequester" type="
              xsd:string" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

7.3. Non-functional requirements

The non-functional requirements are listed in this section. It includes required context, cost and constraints. The XML schema for the elements of the non-functional requirements is shown below.

```
<xsd:element name="nonFunctionalRequirments">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="requiredContext">
        .......
      </xsd:element>
      <xsd:element name="cost">
        .......
      </xsd:element>

      <xsd:element name="constraints">
        .......
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Required Context. The required context lists any conditions or rules the requester will have with respect to the Thing context. Each rule is associated with a priority. Below is the XML schema for defining required context.

```
<xsd:element name="requiredContext">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="rule" type="xsd:string" minOccurs
          ="0"/>
      <xsd:element name="priority" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Cost. Cost information lists the maximum acceptable cost from the requester's point of view. A cost is also associated with a priority. Below is the XML schema for specifying the non-functional requirement cost.

```
<xsd:element name="cost">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:double"/>
      <xsd:element name="currency" type="xsd:string"/>
      <xsd:element name="unit" type="xsd:string"/>
      <xsd:element name="priority" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Constraints. This section lists the trustworthiness and quality guarantees that a requester is requiring from a Thing. It includes interoperability, security, scalability, performance, and availability guarantees. Each required guarantee is associated with a priority. Below is the XML schema for specifying constraints.

```
<xsd:element name="constraints">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="interoperability" minOccurs="0"
          maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="rule" type="xsd:string"/>
            <xsd:element name="priority" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="security" minOccurs="0" maxOccurs
          ="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="rule" type="xsd:string"/>
            <xsd:element name="priority" type="xsd:int"/>
          </xsd:sequence>
```

```
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="scalability" minOccurs="0"
          maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="rule" type="xsd:string"/>
            <xsd:element name="priority" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="performance" minOccurs="0"
          maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="rule" type="xsd:string"/>
            <xsd:element name="priority" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="availability" minOccurs="0"
          maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="rule" type="xsd:string"/>
            <xsd:element name="priority" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

8. Related Work

This research has introduced an SOA based architecture for the publication, and discovery of Things in the Internet of Things. This section is concerned with reviewing related approaches that uses SOA for IoT. Context, scalability, dynamic context, discovery, search, rank, and formality were the criteria used to review the related approaches. Each of these criteria is crucial for any architecture as discussed in Section 3.

In [16], the authors aims to integrate business systems with their manufacturing and logistics processes. This architecture does consider discovery, search, and ranking of Things. This is achieved by using a service catalogue and ranking services based on the program needs during runtime. It also uses web services, which allows it to be scalable. However, this architecture does not consider context at all.

In [11], the authors look at the interaction between IoT and the environment. This is caused by the environment creating events, rather than receiving input from the user. This architecture does allow discovery and search-ability of Things using an IoT Browser but does not have a way to rank them. The browser also allows users to view statistics and certain context of sensors and devices based on current events. The events themselves are viewable and dynamic in nature. This approach is formal, and does show some coding and algorithms. However, this research does not consider scalability.

In [17]. the authors consider IoT Things as resources and builds an architecture for that. Like other approaches, this one does consider discovery by using a service broker, but there is no mention of being able to search or rank these services. Besides discovery, this approach also allows for scalability in its architecture by using the Web of Things. This approach also does not consider any type of context.

In [3], the authors introduce a Discovery Driven SOA. The architecture allows for the discovery of Things, applications, and middleware. However, no mention of being able to rank Things, and only a brief mention of being able to search for Things using a query. By using SOA for services, middleware, computing, storage, and applications, there is support for scalability in this architecture. There was no consideration of context.

In [12], the authors are heavily concerned with dynamic context-awareness through context aware applications. This architecture however, does not allow for any scalability. It also does not allow for any discovery due to its one-to-one behavior.

Table 1 illustrates a summary of our study of the related work. A common theme in all of these architectures is either some type of context awareness and little to no discovery options is provided, or no context with discovery option is provided. None of these architectures support all the comparison criteria. On the other hand, our novel architecture satisfies all of these criteria.

	[16]	[11]	[17]	[3]	[12]
Context	No	Some	No	No	Yes
Scalability	Yes	No	Yes	No	No
Dynamic Context	No	Yes	No	No	Yes
Discovery	Yes	Yes	Yes	Yes	No
Search	Yes	Yes	No	Some	No
Rank	Some	No	No	No	No
Formal	No	Some	No	No	No

Table 1. Related Work

9. Conclusion and Future Work

This paper has introduced an architecture for the publication and discovery of Things in the Internet of Things. To support the architecture, the paper has introduced an architecture for specifying Things and an architecture for specifying queries. In additions two XML languages to support the architecture has been introduced. A prototype has been implemented to simulate the behavior of our architecture. Our future work, include a complete implementation of the architecture while utilizing the scalabiltis of cloud computing to avoid the scalability issue.

References

[1] DEY, A.K. (2001) Understanding and using context. *Personal Ubiquitous Comput.* **5**(1): 4–7. doi:http://dx.doi.org/10.1007/s007790170019.

[2] ERL, T. (2007) SOA *Principles of Service Design* (Upper Saddle River, NJ, USA: Prentice Hall PTR).

[3] GEORGAKOPOULOS, D., JAYARAMAN, P.P., ZHANG, M. and RANJAN, R. (2015) Discovery-driven service oriented iot architecture. In *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*: 142–149.

[4] GEORGAKOPOULOS, D. and PAPAZOGLOU, M.P. (2008) *Service-Oriented Computing* (The MIT Press).

[5] GUBBI, J., BUYYA, R., MARUSIC, S. and PALANISWAMI, M. (2013) Internet of things (iot): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **29**(7): 1645–1660.

[6] HARRIS, R. (2007) Soa done right: the amazon strategy, http://www.zdnet.com/article/soa-done-right-the-amazon-strategy/.

[7] IBRAHIM, N. (2016) A service-oriented architecture for the provision and ranking of student-oriented courses. *ICST Trans. e-Education e-Learning* **3**(9): e2.

[8] IBRAHIM, N., ALAGAR, V. and MOHAMMAD, M. (2014) A context-dependent service model. *EAI Endorsed Trans. Context-aware Syst. & Appl.* **1**(2): e3.

[9] IBRAHIM, N. and BRANDON, B. (2017) Service-oriented architecture for the internet of things. In *The 4th Annual Conference on Computational Science and Computational Intelligence (CSCI'17)*: 1004–1009.

[10] KRAMER, S.D. (2011) The biggest thing amazon got right: The platform, https://gigaom.com/2011/10/12/419-the-biggest-thing-amazon-got-right-the-platform/.

[11] LAN, L., WANG, B., ZHANG, L., SHI, R. and LI, F. (2015) An event-driven service-oriented architecture for the internet of things service execution. *International Journal of Online Engineering* **11**(2): 4–8.

[12] OLSSON, C.M. and HENFRIDSSON, O. (2005) *Designing Context-Aware Interaction: An Action Research Study* (Boston, MA: Springer US), 233–247.

[13] PAPAZOGLOU, M.P. (2008) *Web Services: Principles and Technology* (Prentice Hall), 1st ed.

[14] RAY, P. (2016) A survey on internet of things architectures. *Journal of King Saud University - Computer and Information Sciences* .

[15] SHAMONSKY, D. (2015) Internet of things: Context of use just became more important, https://www.ics.com/blog/internet-things-context-use-just-became-more-important.

[16] SPIESS, P., KARNOUSKOS, S., GUINARD, D., SAVIO, D., BAECKER, O., SOUZA, L.M.S.D. and TRIFA, V. (2009) Soa-based integration of the internet of things in enterprise services. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09 (Washington, DC, USA: IEEE Computer Society): 968–975.

[17] VUJOVIC, V., MAKSIMOVIC, M., KOSMAJAC, D. and PERISIC, B. (2015) Resource: A connection between internet of things and resource-oriented architecture. In *Smart SysTech 2015; European Conference on Smart Objects, Systems and Technologies*: 1–7.

[18] WAN, K. (2006) *Lucx: Lucid Enriched with Context*. Phd thesis, Concordia University, Montreal, Canada.

[19] YU, L., YANG, Y., GU, Y., LIANG, X., LUO, S. and TUNG, F. (2009) Applying context-awareness to service-oriented architecture. In *2009 IEEE International Conference on e-Business Engineering*: 397–402.