Discovering objects and services in context-aware IoT environments

Wei Wang*

School of Engineering and Information Technology,
Murdoch University,
8 Lubberdina Count, Gosnells, WA 6110, Australia
Email: jeffustc@hotmail.com
*Corresponding author

Kevin Lee

College of Arts and Science, School of Science and Technology,
Nottingham Trent University, UK
Email: kevin.lee@ntu.ac.uk

David Murray

School of Engineering and Information Technology,
Murdoch University, Australia
Email: d.murray@murdoch.edu.au

Jian Guo

College of Engineering and Computer Science,
Australian National University, Australia
Email: edison.guo@anu.edu.au

Abstract: The future IoT is expected to connect trillions of objects across infrastructures and domains worldwide. Compared to traditional web applications, the context of objects, such as dynamic object property values, may rapidly change. Browsing web pages with hyperlinks is unsuitable for representing and discovering the dynamic state of objects and services in the IoT, as the dynamic context is less keyword-rich than the relatively static contents on traditional web pages. This paper reviews state-of-the-art in discovering objects and services in the current IoT, and analyses the constraints. This paper also proposes a discovery mechanism, which is extended from previous work GGIoT. The proposed discovery mechanism is aware of dynamic context change in the IoT, and can improve the discovery process using automation.

Keywords: internet of things; IoT; discovery; dynamic property; ontology.

Reference to this paper should be made as follows: Wang, W., Lee, K., Murray, D. and Guo, J. (2019) 'Discovering objects and services in context-aware IoT environments', *Int. J. Services Technology and Management*, Vol. 25, Nos. 3/4, pp.326–347.

Biographical notes: Wei Wang was conferred a Doctor of Information Technology (DIT) degree at Murdoch University, Australia in 2015, a Master of e-Commerce degree at University of South Australia in 2008, and a Bachelor of Computer Science and Technology at University of Science and Technology of China in 2005. In the last five years, he was engaged in some research projects in device integration, data integration, and IoT architecture design.

Kevin Lee is a Senior Lecturer at Nottingham Trent University, UK. He received his BSc, MSc, and PhD from Lancaster University. He was previously a Research Associate at the University of Manchester in the UK, post-graduate Research Fellow at University of Mannheim in Germany and a Senior Lecturer at Murdoch University, Australia. He has published over 60 papers in the areas of distributed systems and adaptive systems.

David Murray is a Lecturer and Academic Chair at the School of Engineering and Information Technology, Murdoch University, Australia. He received his PhD from Murdoch University. He teaches in the areas of data communications and wireless networks. His research is in distributed systems with a focus on data communications. His approaches are usually applied and experimental. He has published over 20 papers in the area of data communications and wireless networks.

Jian Guo is a PhD student in Computer Science at the College of Engineering and Computer Science, Australian National University (ANU). His research interests include machine learning, computer vision. His is also interested in the enabling technologies for the architecture of big data and internet of things.

This paper is a revised and expanded version of a paper entitled 'Building a generic architecture for the internet of things' presented at IEEE Eighth International Conference on the Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, Australia, 2–5 April 2013.

1 Introduction

The *6A connectivity* of the future internet of things (IoT) will enable people and objects to be connected anytime, anyplace, with anything and anyone, using any path/network and any service (ITU, 2005). These virtual objects have identities, attributes, and personalities and can communicate via intelligent interfaces. In the IoT, objects, services and devices can be unpredictably added, moved and terminated, and property values of objects, such as temperature and location, may also change in a few seconds. The spontaneous events will cause dynamic context changes in real-time and worldwide. A great number of heterogeneous devices, such as barcodes, RFID tags and wireless sensors will be attached to physical objects to collect the context distributed across different systems and networks (Gubbi et al., 2013). The future IoT should enable the discovery of the object status in real-time. If many objects or services are discovered, an object should only communicate with certain objects and services to avoid unpredictable interaction. It is beneficial for reducing device and network overhead. With context-awareness, service coordination, composition, optimisation and automation can be enabled in the IoT.

Discovery in dynamic context environments in the IoT is significantly more difficult than searching static contents, such as text and images, on the internet. Browsing static

web pages with hyperlinks is unsuitable for searching objects in the future IoT, as the representation for dynamic state of objects and services is less keyword-rich than contents on traditional web pages (Guinard et al., 2010). In the IoT, collected context can be casual, variable, volatile, redundant, incomplete, inaccurate, and dependent on or conflict with other contexts (Wei and Jin, 2012). Traditional internet applications can use web Services to exchange slowly-changed and unstructured content which are provided and consumed by humans. The current IoT lacks a consistent and structured method of describing entities (Mayer et al., 2012). Discovery in the future IoT require the responses to real-time events and user requests with recently collected sensor data (Yang et al., 2011). Web services can provide device integration and the SOA-based service discovery. However, web services are better for resource-rich devices (Wei and Jin, 2012). The future IoT needs to integrate a great number of devices with limited battery power, processing ability, and memory size, such as passive RFID tags.

In previous work, a global generic architecture (GGIoT) was proposed for the future IoT (Wang et al., 2013). This paper extends the previous work to present a discovery mechanism based on GGIoT. The discovery mechanism aims to enable discovering object, service and other entities in dynamic context environments in the future IoT. This paper is structured as follows: Section 2 discusses architectural requirements for discovering entities in the future IoT. Section 3 reviews related work, and analyses the constraints. Section 4 introduces the previous work GGIoT. Section 5 presents the proposed discovery mechanism in GGIoT. In Section 6, the discovery mechanism is evaluated via case studies. Finally, Section 7 concludes this paper.

2 Architectural requirements

Discovery in the future IoT should not only be used for people, but also for objects, services and applications (Saint-Exupery, 2009). Objects can intelligently search for the ambient objects, services and devices, and interact with them to meet application needs. This section discusses the architectural requirements.

- *Identification:* The identification mechanism in the IoT should enable identifiers to be reconfigured easily (Saint-Exupery, 2009). As most physical objects have a limited lifetime, recycling identifiers is beneficial for reusing the limited identifiers. Because objects are often moved between spaces, identifiers of objects and services need to be independent of current networks (Saint-Exupery, 2009).
- *Decentralisation:* Most objects often interact with other objects in their vicinity and the communication is limited network-wide. It is inefficient to manage trillions of objects and services in a central server due to access latency, processing capability, network traffic, security and privacy issues. To avoid global routing of excessive discovery requests and to reduce device overhead and network traffic, the future IoT should use decentralised discovery topology (Zhang et al., 2011).
- *Service-oriented architecture (SOA):* The SOA can enable seamless interoperability among heterogeneous devices, service optimisation, composition and automation in the IoT. Independent service providers and consumers can publish, and discover services on globally public platforms. Requirements of SOA-based discovery in the

future IoT include minimal service overhead and registration effort, discovering entities in the dynamic context, and on-demand provisioning (Cuinard et al., 2010).

- *Syntactic and semantic coherence:* People may use different languages to describe entities. The discovery mechanisms in the IoT need to interpret exchanged messages among systems. Syntactic coherence can be achieved by setting common data format in exchanged messages (Terziyan et al., 2010). To understand embedded vocabularies in the shared format, semantic coherence can be realised using customised adaptors to interpret the exchanged data, or by building ontologies to regularise semantic rules for describing entities (Terziyan et al., 2010).
- *Automation:* The discovery mechanism in the future IoT should be automated (Mayer et al., 2012). This requires the ability of adjusting discovery strategy based on current context without a manual trigger. Objects can automatically discover and interact with nearby objects. Automated discovery can also facilitate automated service optimisation. Multiple objects and services can intelligently form a composed service on demand.
- *Class-level discovery:* Connecting trillions of devices indicates massive search scope. With the class-level discovery, the scope can be narrowed down to specific types of entities to reduce network traffic for queries. The class-level discovery can be used for coordinating multiple objects in object-to-object (O2O) communication. One type of objects is restricted to interact with pre-defined types of objects and services.
- *Dynamic-context-level discovery:* The future IoT should enable an ability to discover dynamic object property values in real-time. If many eligible objects or services are discovered, an optimal object should be recommended depend on the dynamic context. If no eligible object or services are discovered, on-demand provisioning can be offered by deploying a new service instance to meet application needs (Cuinard et al., 2010).

3 Related work

3.1 Discovery topology

The discovery topologies in the IoT can be classified into four categories: centralised, federal, peer-to-peer and hierarchical. The centralised topology uses a service agent to register services in a local directory. Users can lookup registered services by sending queries to the directory (Zhu et al., 2005). The centralised topology is suitable for discovery in local project scenarios. As the future IoT will reach a global scale, a centralised solution is undesirable (Mayer et al., 2012). The discovery mechanism in a local system may use the private discovery protocols to discover the local devices and services. The federal topology allows multiple systems to create discovery entries for each other in their repositories (Stirbu, 2008). The federal topology is inefficient when many independent systems exist, as excessive adaptors are needed to bridge discovery requests and responds from and to different systems.

A hierarchical topology uses a tree structure to index and discover entities. A system or sensor node has the ability to discover the local resources, and respond to discovery

requests from its parent-system. In the future IoT, objects may move across networks, which dynamically change the topology. The discovery on the tree topology needs to update the topology in real-time. A peer-to-peer topology is a distributed architecture that each system is equally privileged, and can discover each other without previous configuration and mutual agreement (Edwards, 2006). For example, multicast DNS (mDNS) enables devices to find each other and the hosted services network-wide by multicasting messages to the reserved address and UDP port without using third-party servers (Cheshire and Krochmal, 2013). This method is unsuitable for the large-scale scenarios of the future IoT, as it can cause large network-wide interactions. Moreover, embedding an IP stack cannot fit some resource-constrained devices.

3.2 The device-level discovery

Device-level discovery enables devices to register and advertise themselves in IoT networks (Cuinard et al., 2010). If two devices utilise the same Device Discovery Protocols (DDPs), such as the mDNS, they can find each other in networks. Enabling the device-level discovery is a prerequisite of discovering the described objects and services on devices. Service Discovery Protocols (SDPs) are based on DDPs, and are used to advise and discover the hosted services on devices. For example, Bonjour is the SDP of Apple's devices, which uses the mDNS protocols as the underlying DDP (Lee et al., 2007). As device manufacturers may use different SDPs in their products, it is difficult to guarantee that services will interoperate, even though all these devices can communicate with each other using IP protocols (Katasonov et al., 2008).

In the voyager framework, a centralised local server is used to maintain an up-to-date registration of connected devices at the device address level (Savidis and Stephanidis, 2005). When a device connects to networks, it multicasts a HELLO message via UDP. Other devices in a network can detect the device and retrieve metadata of the device from the message. An optimised method can reduce the discovery scope (Guinard et al., 2009). This approach uses device name, type and keyword as filters in a multicast message. Only devices in a network that match the criteria respond to the request. The filters need to be pre-configured on the web-enabled devices.

3.3 Discovering objects and services based on identifiers

In the future IoT, each connected object or service needs to be individually addressed. IPv6 can theoretically provide enough identifiers for connected devices worldwide (Mulligan, 2007). However, embedding IP stacks increases the device cost. Burning serial-numbers into device firmware can be used to identify object-attached devices. However, it is difficult to recycle serial numbers if the objects are abandoned. Both the two identification methods are device-centric, rather than object-centric. The third method is to temporarily allocate identifiers for virtual objects in middleware, which is used by most existing IoT platforms, such as the Xively (2015) platform. In the IoT, identifiers used in different systems can be translated into Uniform Resource Identifiers (URIs). For example, ONS can convert EPC numbers into URIs in EPC Information Services (Mealling, 2004). ID@URI can translate manufacturer-assigned identifiers, such as serial-numbers and barcodes, to URIs of product agents (Främling et al., 2006). Id-IP table is an IoT directory service that can map identifiers of objects, services and operations, into IP addresses (Fortino et al., 2013).

3.4 Discovering objects and services in static context environments

In the IoT, static properties of virtual objects, such as length and name, are constant during the lifetime of a physical object. Interface descriptions of virtual services are static, but input and output of the services can be dynamically changed. Discovering objects and services in the static context environments is similar to service discovery in traditional web services. Static properties of objects and services can be defined as structured metadata, and self-described APIs of virtual objects and services can be published for discovery. As people may use different words to describe entities, it is difficult to standardise metadata to describe trillions of objects in the future IoT.

Three models are used to map discovery requests into metadata from device messages. The top-to-bottom method emphasis on application needs, but does not address data representation in device messages. For instance, goal-based service framework (GSF) decomposes a discovery request into several sub-tasks for discovering the requested services (Santos et al., 2009). GSF does not address how to describe objects in sensor messages and how the message can be interpreted. The bottom-to-up model focuses on representation of objects and services in device messages, but does not consider how third-parties can discover required objects and services. For example, a service provider module on sensors can offer self-described metadata used for describing the attached objects (Fortino et al., 2013). This method requires pulling metadata from devices via web services, and users need to represent data in sensor messages by following required rules, which hinder the efficiency on a large scale of the future IoT.

In the end-to-middle model, discovery requests in different applications are mapped into metadata of connected devices. For instance, in the aura framework, a service supplier abstracts services from devices, and an Environment Manager in gateways maps application requests into the abstracted services based on the collected context (Sousa and Garlan, 2002). Implementation of the framework is challenging, as mapping contexts between raw sensor data and application requests involves subjectivity. Another method uses human operators to map contexts of sensor data into application requests (Ostermaier et al., 2010), as people have much better context-awareness. This method would significantly increases cost, and hinder the automation.

3.5 Discovering objects and services in dynamic context environments

To enable discovery in dynamic context environments in the IoT, three requirements need to be fulfilled. First, the latest status of objects and services needs to be updated in real-time. Second, consistent methods are required for descripting entities. Finally, if multiple eligible objects or services are discovered, the future IoT should have the ability to select an optimal one depending on the dynamic contexts. Most existing IoT platforms require connected sensors to embed RESTful web services. The requests are initiated by the IoT platforms via HTTP operations, and sensors passively respond (Rodríguez-Domínguez et al., 2012). The pull-style communication allows the IoT platforms to retrieve messages from sensors on demand. As the dynamic status of an object may be requested by many applications simultaneously in the future IoT, the pull-style would exhaust energy of a sensor device if too many requests are processed. Furthermore, this method also adds hardware requirements for the connected devices.

To save sensor energy, an optimised method can periodically retrieve messages from sensors using a time window (Ostermaier et al., 2010). However, if a window length is

defined improperly, dynamic object status cannot be updated in some time-sensitive applications. The APPUB technique utilises an adaptive time window to cache sensor messages via RESTful web services (Butt et al., 2013). An adaptive timer can vary the length of the time window to request data from sensors. If a sensor is frequently requested, the window length is decreased. This optimal method aims to balance the trade-off between device energy efficiency and request frequency, while real-time status of objects is not considered. Due to inconsistent description of entities across systems, reasoning technologies are used to convert sensor messages to the required representation via adaptors (Barbero et al., 2011). An adaptor is only used for limited types of data conversion and also failure-prone.

To select an optimal object or service, all search results need to be compared based on specified criteria. Thus, dynamic context of all involved objects and services need to be analysed. However, multicasting discovery requests to all sensors in networks add network traffic and energy consumption of sensors. A compromised method compares historical sensor data in databases (Elahi et al., 2009). By estimating a probability that all involved sensors could output the sought property value at the discovery time, the involved sensors can be ranked in a descending order. The first sensor has the highest probability to match the discovery query. A search engine retrieves real-time messages from the first sensor. If the first sensor cannot match the request, the search engine retrieves messages from other sensors in the ranked order (Elahi et al., 2009). This method uses historical data in databases to estimate possible property values in sensor messages. It can find an eligible object based on specified dynamic property values. However, the discovered object may be suboptimal, as the probability ranking is ordered based on historical behaviour of objects, rather than real-time status of objects.

3.6 Discovering entity classification in semantic models

Building semantic models, such as ontologies, can describe classification of entities in the IoT. For example, sensor network ontology can describe types of sensors and sensor networks, and provide methods of using ontologies to develop applications (W3C, 2011). To classify objects and services in the IoT, a service ontology use three sub-ontologies to describe hosted services in sensors, sensor locations, and physical properties (Kim et al., 2008). However, as data schemes and entity properties cannot be customised by third-parties, the ontology has limited description ability and lacks scalability to support emerging resources. In the IoT-A project, the class information model is used to classify fine-grained entities and build complex relations between the entities (Walewski, 2011). This model does not specify how entities are represented in sensor messages and how context is abstracted and interpreted.

3.7 The SOA-based discovery

In traditional internet applications, the implementation of SOA is mostly based on the WS-* specification. In the IoT, to reduce the overhead caused by SOAP and XML, device profile for web services (DPWS) is a subset of the WS-* specification that enable

wireless sensors to publish and discover web services by exchanging service description network-wide (Guinard et al., 2012). Compared to the WS-* specification, RESTful web services can reduce communication overhead. However, RESTful web services focus on operating sensor data via URIs, rather than providing off-the-shelf services to third-parties (Guinard et al., 2009).

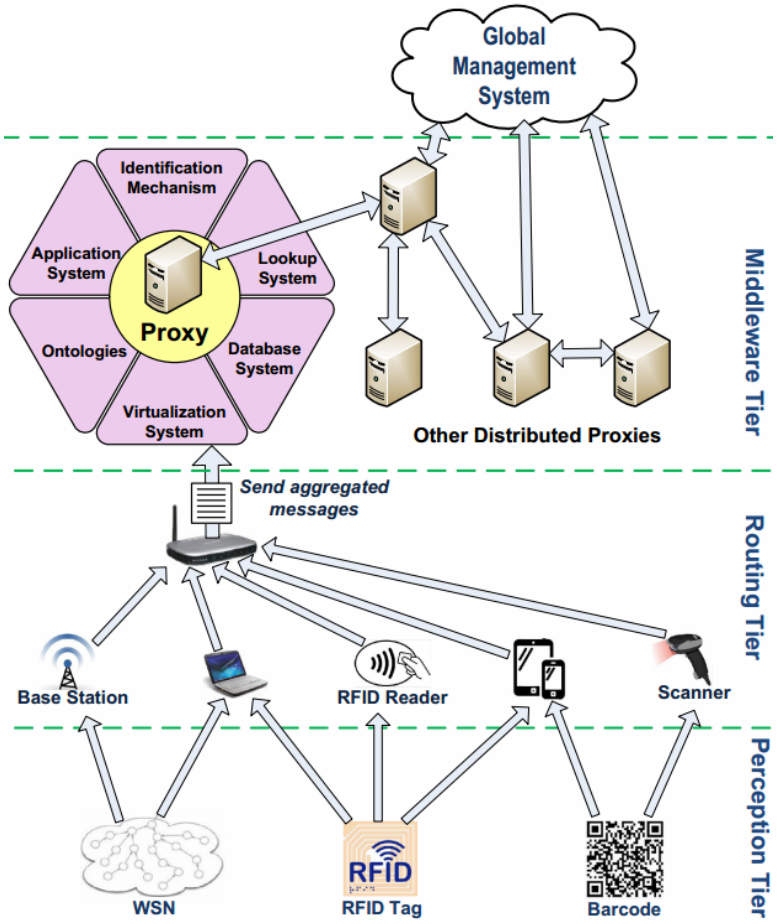
Component-based middleware can also use the SOA to publish and discover services. Flissi et al. (2005) propose a discovery mechanism for component-based middleware. Service registry (SR) is a component that can register, update and remove services on the server-side. On the client-side, such as wireless sensors, service activator (SA) is a component used to interact with the SR. The SA can discover registered services in the SR, and publish their services to the SR (Flissi et al., 2005). The SA component is bound to a service component, and plays a role of API for the service component. It is similar to the SOA principle in the WS-* specification.

Web services have been given priority to be used in the current IoT, as standards have been widely accepted in internet applications. Using human-readable web protocols allows people to easily develop, discover, understand and access services. Component-based middleware, such as EJB and CORBA, outperform Web Services (Petritsch, 2005). Performance is enhanced by using binary protocols in communication, which reduce transmission size of messages. As most IoT services are based on O2O communication, using machine-readable binary protocols can improve performance, and reduce cost and size of object-attached devices.

4 Previous work

In this paper, the proposed discovery mechanism is based on the previous work which presented a GGIoT for the future IoT (Wang et al., 2013). GGIoT is binary-protocol-enabled and component-based. It is independent of specific devices, platforms, systems and networks. GGIoT can efficiently integrate resources across domains and parties. Figure 1 shows the overall architecture of GGIoT, which includes: perception, routing, middleware and global management system (GSM) tier. The perception tier collects raw data from the physical world using different devices, such as sensors and RFID tags, to represent the data as dynamic property values for virtual objects in middleware. A sensor message consists of a system-allocated object ID and dynamic property values of the physical object. For example, in a sensor message '*MFS003412 16*', '*MFS003412*' is a system-allocated object ID and '*16*' represents a dynamic object property value of 'temperature'. To reduce message size to fit resource-constrained devices, other elements such as data schema, description of object properties, measure of units and static property values, are pre-described in object templates in ontologies. By mapping sensor messages with the related object templates in proxies, message meaning can be interpreted. As objects are described in shared templates rather than in device messages, meaning of object description can be globally consistent. As barcode and passive RFID tags do not have sensing ability, a message only contains an object ID links to a static virtual object in middleware.

Figure 1 The overall architecture of GGIoT (see online version for colours)



The routing tier builds communication channels between a variety of end devices and middleware in proxies. Intermediate devices, such as smart phones and laptops can be used to relay received messages from end devices to a gateway. The gateways route the aggregated messages to the middleware tier. At the middleware tier, a distributed proxy consists of: identification mechanism, ontologies, virtualisation system, lookup system, application system and database system. The identification mechanism can assign a global-unique ID for a connected object. The allocated object ID is formatted into output messages of an object-attached device. In middleware, the physical object is virtualised as an object component. The related services are virtualised as service components that can receive and process messages from the object component. When the object component is initialised in the virtualisation system, the component ID is identical to the pre-allocated object ID in the device messages. The aim of this design is to use introspection capability of component-based middleware to discover virtual objects based on system-allocated component IDs. These identifiers are temporarily allocated to virtual objects and services, and can be recycled and reused.

In the ontologies, a generic template is used to describe a primitive type of entities, such as object, service and unit of measure (Wang et al., 2013). Each template can be accessed by a URI. By adding new property elements into a generic object template, a new object template can be generated to describe a customised object, and a new URI is allocated for the customised template. In the service ontology, a service template is used to generate one type of virtual services. Similarly, existing service templates can also be customised by users. Other entities, such as device, time, location, can also be represented in the ontologies. To adapt to new entities, the ontologies can be updated. To maintain global consistency, all ontology data is managed by the GMS. Other distributed proxies periodically download updates from the GMS. The GMS assigns a range of identifiers to each distributed proxy. Then the distributed proxies can further allocate and recycle the identifiers to and from the hosted virtual objects and services.

In the IoT, most O2O communications occur within specific scopes. Using distributed proxies to coordinate local O2O communication can reduce network traffic and access latency. The distributed proxies can run on a local network, metropolitan area network, or in the cloud. Location and specification of a proxy is determined by the connected objects and services, required resources and application requirements. All distributed proxies form a mesh network worldwide. If O2O communication is beyond the range of a distributed proxy, the GMS can coordinate the communication.

An object component can receive real-time messages from an object-attached device, and relay the messages to the wired service components. By mapping the received messages with the associated object templates, the service components can interpret the dynamic status of the objects, and perform required data processing tasks, such as displaying, routing, aggregating, filtering, storing, monitoring and converting. A service component has an interface to output data, and more than one receptacle to receive messages from the wired object components. Thus, a service component can coordinate O2O communication of many objects. To avoid unpredictable interaction among objects, it is unnecessary to trigger O2O communication if no services can be provided to two objects. The constraints are defined in object and service templates.

The application system can offer various application development tools for third-party users. For example, template editing tools can be plugged into web browsers. Thus, users are allowed to describe objects and services in templates rather than in device messages. To save development cost, tools can be used for composing many atomic services into a coarse-grained service. Tools can also be provided to non-expert users for automatically formatting output messages of devices as required. Thus, developers can focus on designing applications without concern for the underlying infrastructure and devices. Users are also allowed to consume IoT services in a straightforward way.

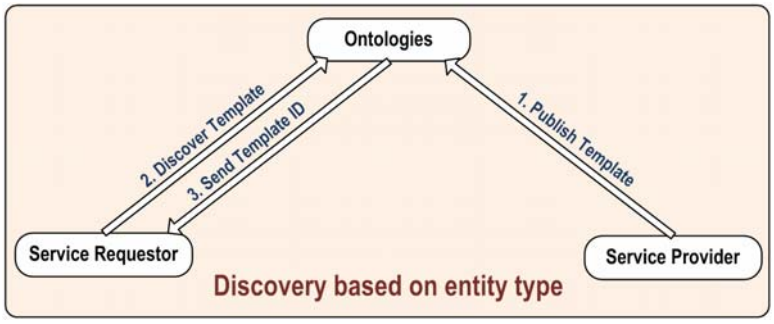
5 The discovery mechanism in GGIoT

The lookup system offers the proposed discovery mechanism in GGIoT. A distributed proxy is responsible for publishing, indexing and discovering local resources. When a proxy cannot find a requested resource, the query will be routed to a destination proxy via coordination of the GMS. The discovery mechanism includes five basic types of discovery which are not only used for people, but also for objects and services.

5.1 Discovery based on entity type

GGIoT clearly separates service type and instance. A template in ontologies describes one type of virtual object or service. The associated components in middleware are virtualised instances to implement the virtual objects or services. Figure 2 shows the discovery based on entity type. To find an existing template to describe an object type, users can specify a template ID, or use keywords, such as object properties, to look for a suitable template. Then the lookup system discovers a list of template candidates which can be translated into different languages depending on users' preferences. The translation is convenient for users to discover and view object templates in their own language. Then users select the most suitable template to fit the new connected object. If a ready-made template cannot be found, users can customise an object template by editing an existing template. When a component is generated in a distributed proxy, an Identifier Manager in the proxy registers the component with the related template ID. Thus, a virtual object or service does not need an interface component to describe the hosted service. Metadata of the virtual object or service can be discovered via looking up the template in the ontologies.

Figure 2 Discovery based on entity type (see online version for colours)



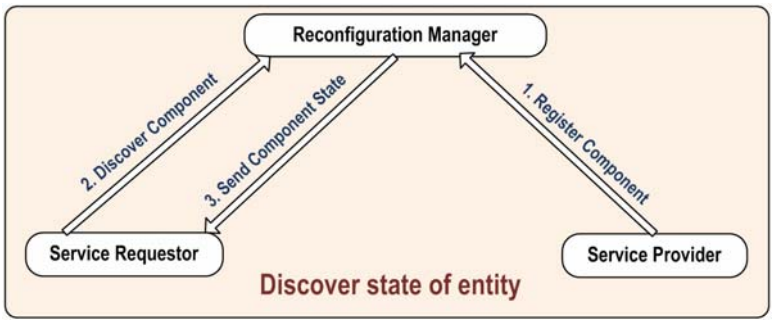
5.2 Discovery of entity states

In GGIoT, an object component ID is identical to the object ID in the device messages. A component ID is used to address a virtual object in middleware. Component states, such as wired, activated, deactivated and deleted, represent states of the object, such as connected, available, suspended, and terminated. If an object component has not received messages from the associated object above a predefined period of time, the object component can be switched to suspended state. If the object is reconnected, the component is reactivated. The temporary disconnections may be caused by different reasons, such as turning off devices and network failures. If the inactive state lasts longer, the object component is removed in middleware to release system resources.

Figure 3 illustrates discovery of entity state in GGIoT. The Reconfiguration Manager can register, inspect and reconfigure components in component-based middleware. When a component is generated in middleware, it is registered in the reconfiguration manager. By invoking a set of control commands, a component can be reconfigured on runtime, such as activated, suspended, terminated, wired and unwired with other components. The

reconfiguration manager has the inherent ability to lookup state of the registered components, such as component ID, wiring and activation.

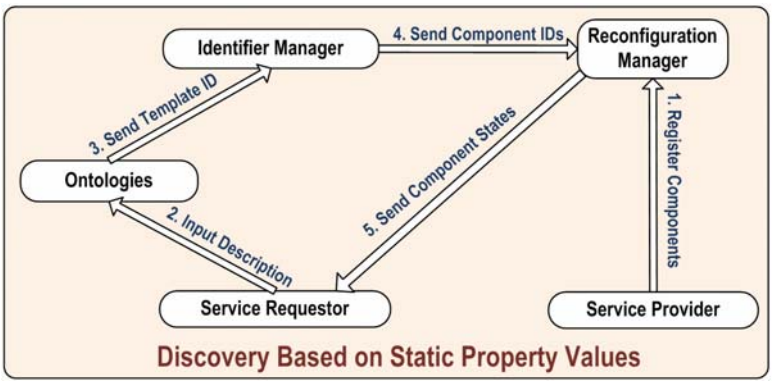
Figure 3 Discovery of entity state (see online version for colours)



5.3 Discovering objects and services in static context environments

In GGIoT, static object properties, such as '*Length*', and the values are described in object templates. The association between an object component and the template is registered in the Identifier Manager of a proxy. Thus, the proxy has the ability to discover the static context of all registered objects and services. Figure 4 illustrates the discovery of object and service instances based on static properties. In a proxy, a search engine in the ontologies can analyse user-described keywords, and search for all eligible templates from the pre-registered templates in the Identifier Manager. The discovered templates represent types of eligible virtual objects hosted in the proxy.

Figure 4 Discovering object and service instance based on static property values (see online version for colours)



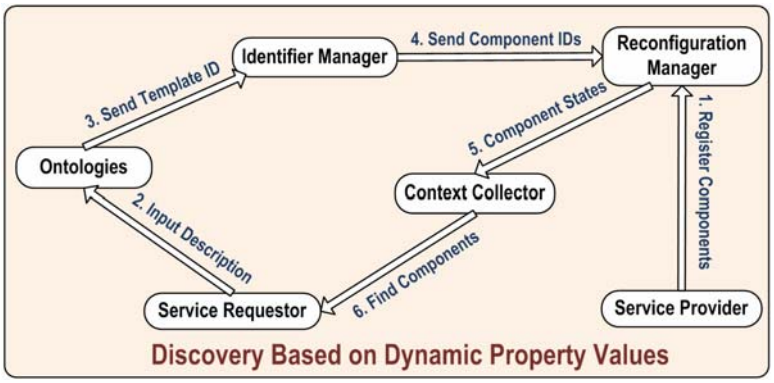
By looking up the associated object component ID, the Identifier Manager can find all eligible object components hosted in the proxy. Within all the discovered components, only the object components in activated state can meet the application requirements. The Reconfiguration Manager queries the states of all discovered eligible components, and then further filter out the activated object components. This process demonstrates the discovery of object and service instances based on static properties and property values.

As the static context of virtual objects is in the templates, the process does not need to interpret sensor messages. The discovery scope is narrowed down in an order of object type, object instance and object states.

5.4 Discovering objects and services in dynamic context environments

In GGIoT, dynamic object properties, such as '*Location*' and '*Temperature*' can also be statically represented in the associated templates. Discovering object and service instances based on the dynamic properties is also in the static context environments. However, dynamic object property values can only be retrieved from real-time sensor messages. Figure 5 shows the discovery of entities in dynamic context environments.

Figure 5 Discovering object and service based on static property values (see online version for colours)

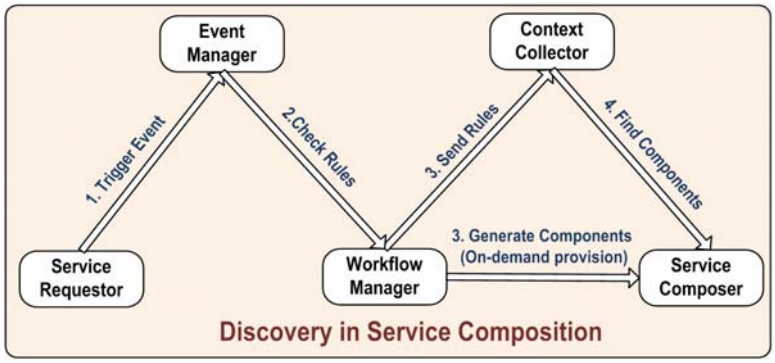


Comparing to the discovery in static context environments, a component of Context Collector is added, which is used to collect dynamic context of all involved object components. The Context Collector is wired to all the object components to receive real-time messages of the objects. By mapping the messages of the virtual objects into the associated templates, dynamic property values of these objects can be abstracted and interpreted. By comparing the searched property value with the collected dynamic property values, the Context Collector is able to discover all eligible virtual objects in a proxy. The discovery scope is narrowed down in an order of object type, object instance, object state, and dynamic property value.

5.5 Discovery in service composition

In GGIoT, multiple object and service components can be automatically combined to deliver a composed service on demand. As dynamic property values of virtual objects and services keep changing, the discovery mechanism needs to find the most suitable objects and services to compose a service in dynamic context environments. Figure 6 illustrates an event-triggered discovery in a service composition. Discovery requests are initiated by events rather than queries from human users.

Figure 6 Discovery in service composition (see online version for colours)



When a component is initialised, activated, suspended or removed, events are reported to the Event Manager. Thus, state changes of a virtual object or service are notified as real-time events in application workflows. Property value changes of a virtual object or service can also be used to trigger a predefined event. For example, in the previous work, if the monitored temperature values are above the threshold value, a notification is triggered (Wang et al., 2013). A triggered event is sent to Workflow Manager that checks if the event is registered in a workflow of a composed service. If a workflow exists, a Context Collector is generated to collect dynamic context from all involved components, and find all available components that can join the service composition. The discovered component IDs are sent to a component of Service Composer. The Service Composer implements the service composition by wiring and unwiring these object and service components on demand. If multiple eligible components exist, the Service Composer looks for an optimal one based on the dynamic context. An event can also trigger other actions, such as generating a service component in a workflow.

6 Evaluation

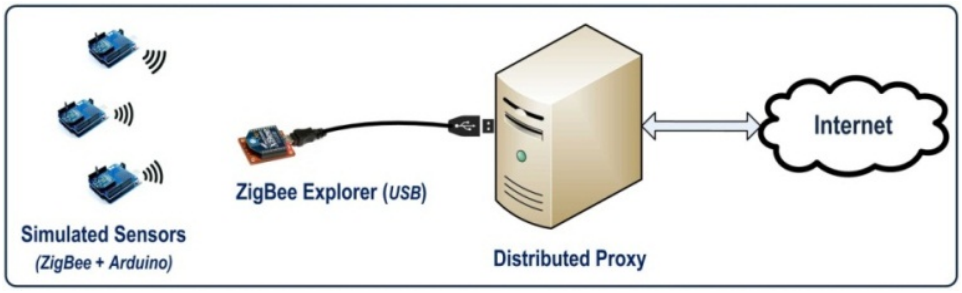
Previous work has evaluated the process of virtualising physical objects and services into middleware components, and O2O communication among the objects in GGIoT (Wang et al., 2013). This section further evaluates the proposed discovery mechanism. Two proof-of-concept case studies are presented to illustrate the discovery in dynamic context environments and discovery optimisation. In middleware, interactions among the virtual objects and services are represented as bindings between the components. All components can be reconfigured at runtime to adapt to context changes.

6.1 Evaluation setup

As GGIoT is independent of specific devices, systems, networks and applications, the proposed discovery mechanism are not confined to the setup and case studies in this section. In this test, an Arduino UNO and a Zigbee Xbee module were composed to simulate a generic sensor or RFID tag. Formats of device messages were written into an Arduino UNO (Arduino, 2014), and transmitted via a ZigbeeXbee (ZigBeeAlliance, 2015). Figure 7 shows the evaluation setup. A dual-core PC acts a distributed proxy. The

simulated sensors transmit messages of involved objects to the proxy. A ZigBee Xbee Explorer aggregates messages of all sensors in a WSN, and send the messages to the proxy via a USB port. The proxy runs the designed virtualisation system based on the LooCI OSGI V1.0 (Hughes et al., 2009). Windows XP are used as underlying systems. XML was used as format to describe ontologies and templates in the proxy. The virtualisation system can interpret meaning of the device messages by mapping the messages into the relevant templates. To reduce processing load for a distributed proxy, only parts of object property values are interpreted in a specific service. For example, it is unnecessary to interpret '*Length*' in a temperature monitoring service.

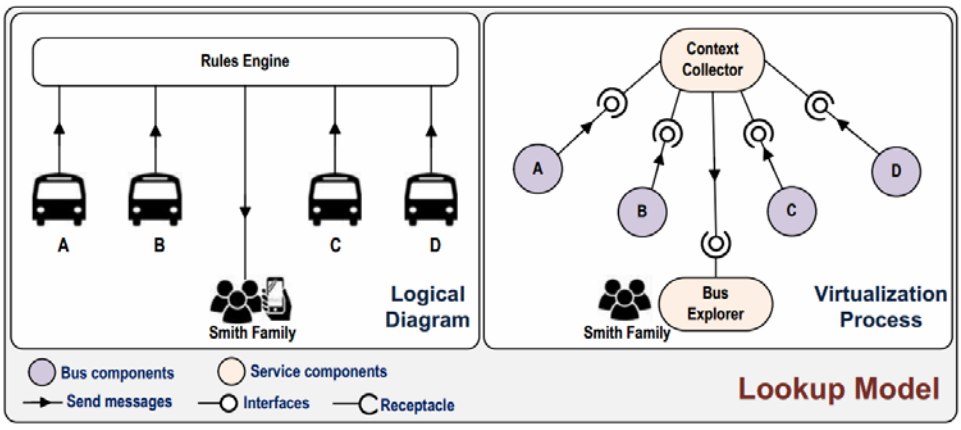
Figure 7 The evaluation setup (see online version for colours)



6.2 *Discovery in dynamic context environments*

In Figure 8, this case study assumes that a bus company provides a service that allows users to lookup vacant seats of the buses via mobile phones. The Smith family tries to discover an available bus to a hotel in a foreign city. They send a discovery request to a near proxy by a mobile phone. The discover request includes the required number of vacant seats and specified hotel address. By analysing the request, a search engine in middleware generates a lookup service that can dynamically discover suitable buses with enough vacant seats for the Smith family. Then the discovered results are sent to the users' mobile phone. A discovery result may contain multiple available buses.

Figure 8 The lookup model in GGIoT (see online version for colours)



Each seat on a bus is equipped with a pressure sensor to sense if a seat is taken by a passenger. By accumulating a total number of vacant seats in the bus, the bus can generate a dynamic object property value '*Vacant-seats*' in real-time. As passengers may get in and out of the buses without prediction, property values of '*Vacant-seats*' are dynamically changed. Via mobile networks, the bus can route the messages to the object component in middleware of a distributed proxy. Each message contains two data fields 'ObjectID' and '*Vacant-seats*'. Other static bus properties and the values are pre-described in the related object templates in ontologies. By wiring with all the involved bus components, the Context Collector can receive messages from these bus components, and interpret dynamic property values of 'Vacant-seats' of these buses. By comparing the values of 'Vacant-seats' to the user-requested number of vacant seats, all eligible buses with enough vacant seats can be discovered. A Bus Explorer on the users' phone is used to input discovery requests and receive the search results.

Figure 9 The workflow of object discovery (see online version for colours)

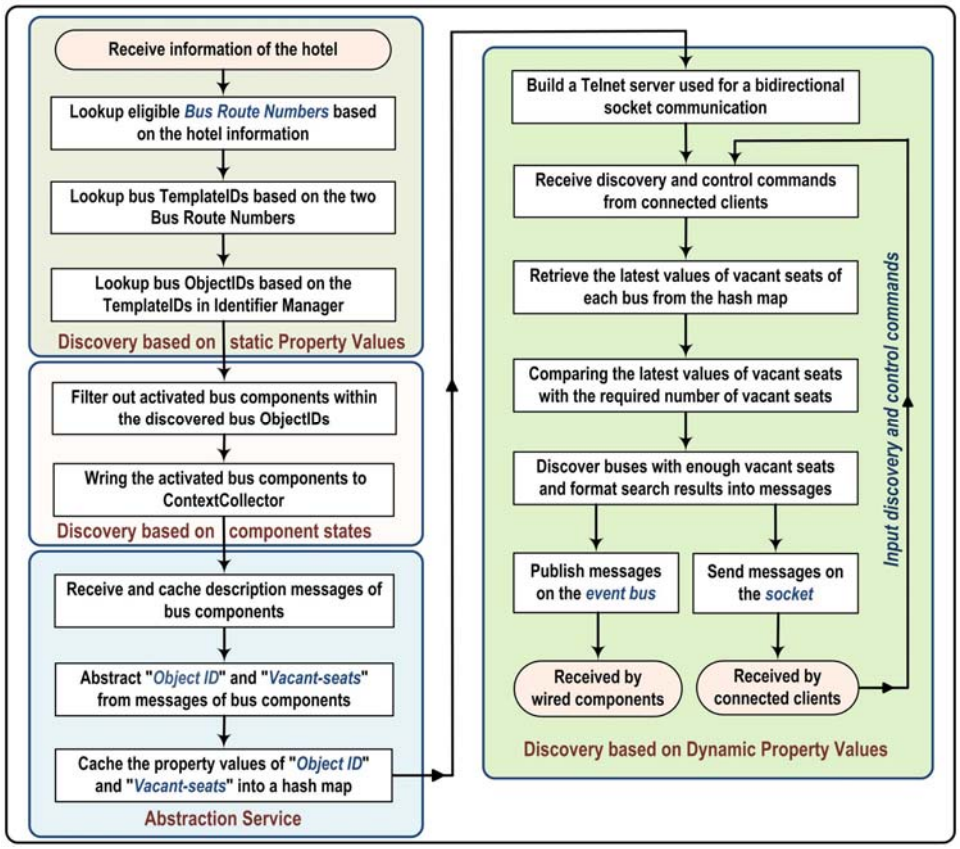


Figure 9 illustrates the workflow of the object discovery. The module *Discovery based on Static Property Values* performs the discovery of an object instance based on object type. This test assumes that *Bus Route Number* is a static property of all the involved buses, and buses in two route numbers *R0098* and *R0564* pass the hotel. Multiple buses may run on the same route. By parsing the object templates in the ontologies, template IDs of all

bus components on the two bus routes can be found. In the proxy, as the Identifier Manager has pre-registered the mapping between the hosted virtual objects and the associated template IDs, object IDs of all bus components on the two bus routers can be queried. The module *Discovery based on component states* filters out all working buses that stop at the hotel. In middleware, a bus component has states, such as activated or suspended, to represent if the bus is in service. By querying states of the bus components, activated bus components on the two bus routes are selected.

To collect dynamic context of the selected buses, the Context Collector is wired to the activated bus components to receive the real-time messages. The module *Abstraction Service* abstracts object IDs and dynamic values of '*Vacant-seat*' from the messages. The mappings between the object IDs and values of '*Vacant-seats*' are cached in a hash map, and the hash map is updated when new bus messages arrive. It provides a repository to cache the latest property values of '*Vacant-seat*' of these buses in this discovery. If O2O communications across many networks, two methods can be used to verify the latest messages. For sensors with accurate timing capability, timestamps can be added into output messages. Otherwise, timestamps can be appended to sensor messages at a local proxy when the proxy receives the messages.

The module of *Discovery based on dynamic Property Values* builds a Telnet server in the Context Collector to receive and to analyse user-inputted requirements. A Telnet client is running in the Bus Explorer comment. By wiring the Context Collector to the Bus Explorer, a user can send a required number of seats to the Context Collector. The Telnet server abstracts the required number of seats, and then stores the value into a variable *requiredSeats*. A service periodically retrieves the cached messages from the hash map every 5 seconds. By comparing the latest values of '*Vacant-seats*' of the buses with the variable *requiredSeats*, all buses with enough seats are discovered in real-time. Search results are formatted into messages and then sent to the connected Telnet clients. In this test, by inputting different values of *required number of seats*, the Context Collector returned all available buses and the number of vacant seats.

This test outputs search results via two methods: Telnet-based and middleware-based. As component-based middleware runs in closed environments, it cannot interact with other middleware or Web Services. To resolve this issue, the Telnet-based method enables isolated middleware components to output messages on sockets. Thus, users on other platforms can also interact with the discovery service by connecting to the socket. For example, the user interface can be designed as a RESTful API. For the component-based method, search results can be converted into required data formats, and published on the event bus of the middleware. Thus, other service components that subscribed to the Context Collector can receive and process the search results.

6.3 Discovery optimisation

In some situations, many eligible objects and services are discovered to respond to the same request. The discovery mechanism in GGIoT can recommend an optimal one from all searched results depending on the dynamic context. In Figure 10, the Smith family wants to add other search condition, discovering the nearest bus with enough vacant seats, to optimise discovery results. This scenario assumes that both bus A and B can provide enough vacant seats, and the Smith family starts to wait for bus A in a bus stop. They use a Bus Tracker service running on a mobile phone to track real-time position of bus A. When bus A stops at a near bus stop, bus B overtakes bus A and then bus B

becomes the nearest bus. The Context Collector is aware of the context change. It disconnects the coupling between bus A component and the mobile phone, and then connects bus B component with the phone. Consequently, the Bus Tracker component on the phone is switched to receive real-time messages from bus B.

Figure 10 A case study based on the interaction model in GGIoT (see online version for colours)

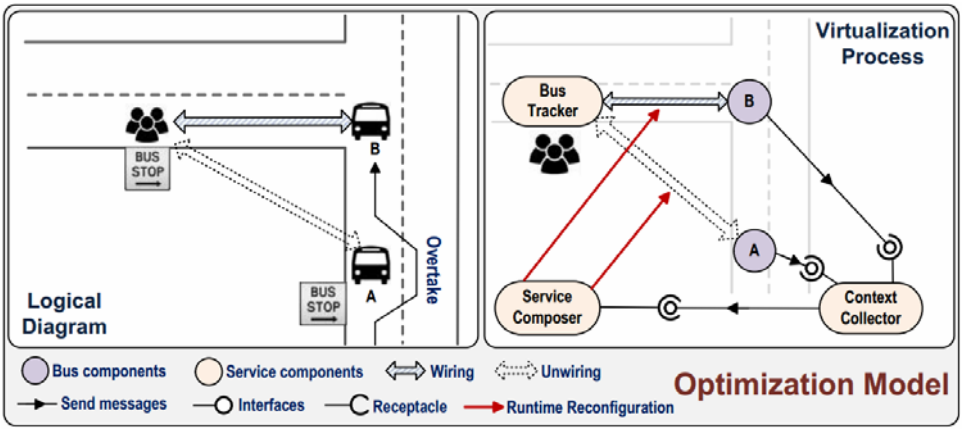
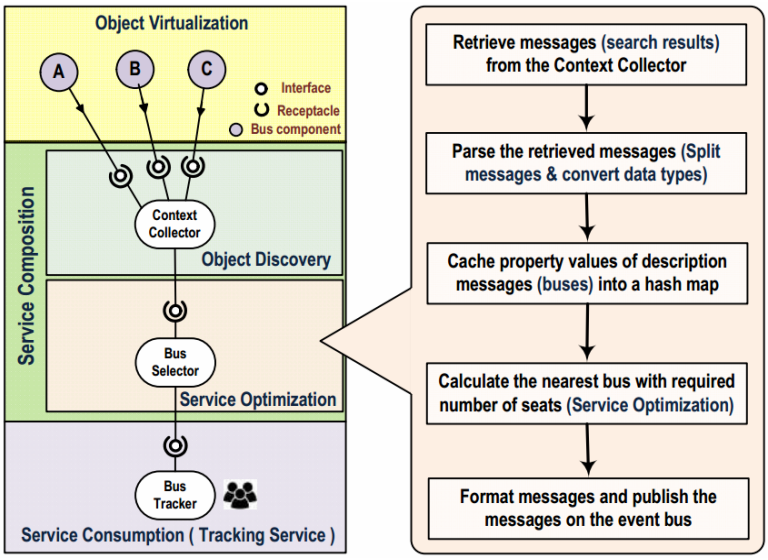


Figure 11 The workflow of service optimisation (see online version for colours)



In the test, a Service Composer is wired to the Context Collector to receive real-time messages of all discovered bus with enough vacant seats. Thus, the Service Composer is aware of context change of all the eligible buses. When the event '*Bus B overtakes bus A*' occurs, the Service Composer triggers a predefined action to discover an alternative bus component to replace of the bus A component. By analysing collected contexts, bus B component is discovered. Then the Service Composer coordinates the communication by unwiring the Bus Tracker and bus A component, and connecting the Bus Tracker to bus

B component. Figure 11 shows the relation between the two case studies, and a workflow of the service optimisation scenario. A component of Bus Selector was designed. The Bus Selector can dynamically determines the nearest bus with required number of seats, and then route the messages of the nearest bus to the Bus Tracker.

By wiring to the Context Collector, the Bus Selector can receive real-time messages of all the discovered buses with required number of seats. Each received message contains multiple dynamic object property values, such as *Vacant-seats* and *Location*. By default, data type of all received messages is String. It is necessary to split all data elements of a message and convent these data elements to required data types. For example, in this case, to calculate the minimal distance between a bus and the user, property values of 'location' were tuned into Float. In GGIoT, data types of property values are described in the object and service templates.

All the converted data elements of a message are stored in an object of *busProperty* which is cached in a hash map. When the Bus Selector receives a new message from the Context Collector, a method of *findNearestBusId()* calculates a real-time distance between the bus described in the message and the user. By comparing the distance to the previously-stored minimum distance, the method decides if it updates the stored object ID of the nearest bus and the minimal distance. In this case, the Bus Selector acts as a service optimiser to recommend a bus with the minimum distance from all available buses with enough number of seats. The Context Collector and Bus Selector offer a composed service to the Bus Tracker. *Number of vacant seats* and *Bus location* are two types of collected dynamic contexts. The two services can also be designed in one service component to reduce the communication overhead between them, with the cost of removing the loose-coupling between the two service components.

7 Conclusions

The current IoT is constrained to specific devices, platforms, domains and services. Many obstacles, such as incompatible existing standards, lack of applications, high cost and difficult deployment, hinder widespread use. Due to the inconsistent object representation in device messages worldwide, it is challenging to design a discovery mechanism across platforms, systems, domains and applications. WEB services can enable discovery based on the SOA principle, and has been given priority to be used in the current IoT, as the standards have been widely accepted in traditional internet applications. However, human-readable protocols add a non-negligible overhead for a mass of O2O communication and discovery queries in the future IoT. Moreover, using web services also increases cost and size of the embedded devices.

For the RESTful web services, the pull-style discovery queries would exhaust energy of a sensor if the sensor is requested by too many applications. In a proxy, using time windows to request and cache messages of web-enabled devices can save energy used for frequent requests. However, this method may not discover the real-time state of a connected object. Moreover, multicasting discovery requests network-wide can cause random interaction among objects when no service is provided to the objects. As the current IoT cannot clearly separate service type and service instance, more data need to be transmitted and processed, and the discovery scope is significantly increased.

This paper extends the previous work, and presents a discovery mechanism based on the architecture GGIoT. To reduce device overhead and network traffic, architecture of

GGIoT is de-centralised; a distributed proxy is only responsible for discovering the local objects and services. As GGIoT uses binary protocols for O2O communication, most resource-constrained devices, such as passive RFID tags, can be integrated. To avoid multicasting discovery queries to all devices network-wide, connected physical objects are virtualised as object components in middleware. Thus, discovery requests are sent to the Context Collector which can collect dynamic context from the involved object components as required.

The proposed discovery mechanism enables discovery of entities in dynamic context environments in the future IoT. The discovery scope is gradually narrowed down in order of object type, object instance, object state, and dynamic property values. Thus, only messages of involved objects and services are collected and interpreted in a discovery request. A local proxy coordinates interaction among multiple objects and services based on object and service type. Thus, random interaction between objects can be avoided. If many eligible objects or services are discovered in a request, the discovery mechanism can intelligently select an optimal object or service to meet user requirements. The Service Composer looks for the most suitable objects and services to build a composed service to deduce development cost. The constituent objects and services in the composed service can be dynamically replaced by other virtual objects and services, due to the dynamic context change in a specific application.

References

Arduino (2014) *Arduino Uno Overview* [online] http://arduino.cc/en/Main/arduinoBoardUno (accessed 10 December 2014).

Barbero, C., Dal Zovo, P. and Gobbi, B. (2011) 'A flexible context aware reasoning approach for iot applications', *12th IEEE International Conference on Mobile Data Management (MDM)*, Vol. 1, pp.266–275, doi: 10.1109/MDM.2011.55.

Butt, T.A., Phillips, I., Guan, L. and Oikonomou, G. (2013) 'Adaptive and context-aware service discovery for the internet of things', *Internet of Things, Smart Spaces, and Next Generation Networking*, pp.36–47, DOI: 10.1007/978-3-642-40316-3_4, Springer.

Cheshire, S. and Krochmal, M. (2013) 'RFC 6762: multicast DNS', *Internet Engineering Task Force (IETF) Standard*, DOI: 10.17487/RFC6762.

Cuinard, D., Trifa, V., Karnouskos, S., Spiess, P. and Savio, D. (2010) 'Interacting with the SOA-based internet of things: discovery, query, selection, and on-demand provisioning of web services', *IEEE Transactions on Services Computing*, Vol. 3, No. 3, pp.223–235, DOI: 10.1109/TSC.2010.3.

Edwards, W.K. (2006) 'Discovery systems in ubiquitous computing', *Pervasive Computing*, Vol. 5, No. 2, pp.70–77, DOI:10.1109/MPRV.2006.28, IEEE.

Elahi, B.M., Romer, K., Ostermaier, B., Fahrmair, M. and Kellerer, W. (2009) 'Sensor ranking: a primitive for efficient content-based sensor search', *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pp.217–228.

Flissi, A., Gransart, C. and Merle, P. (2005) 'A service discovery and automatic deployment component-based software infrastructure for ubiquitous computing', *Ubiquitous Mobile Information and Collaboration Systems (UMICS 2005)*, pp.601–615.

Fortino, G., Lackovic, M., Russo, W. and Trunfio, P. (2013) 'A discovery service for smart objects over an agent-based middleware internet and distributed computing systems', *International Conference on Internet and Distributed Computing Systems*, pp.281–293, DOI:10.1007/978-3-642-41428-2_23, Springer, Berlin Heidelberg.

Främling, K., Harrison, M. and Brusey, J. (2006) 'Globally unique product identifiers-requirements and solutions to product lifecycle management', *Proceedings of 12th IFAC Symposium on INCOM*, Vol. 39, No. 3, pp.855–860, DOI:10.3182/20060517-3-FR-2903.00399.

Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M. (2013) 'Internet of things (IoT): a vision, architectural elements, and future directions', *Future Generation Computer Systems*, Vol. 29, No. 7, pp.1645–1660, DOI:10.1016/j.future.2013.01.010.

Guinard, D., Ion, I. and Mayer, S. (2012) 'In search of an internet of things service architecture: REST or WS-*? A developers' perspective', *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pp.326–337, Springer, Berlin Heidelberg.

Guinard, D., Trifa, V, and Wilde, E. (2010) 'A resource oriented architecture for the web of things', *The Internet of Things (IOT)*, pp.1–8, DOI: 10.1109/IOT.2010.5678452.

Guinard, D., Trifa, V., Spiess, P., Dober, B. and Karnouskos, S. (2009) 'Discovery and on-demand provisioning of real-world web services', *IEEE International Conference on Web Services*, DOI: 10.1109/ICWS.2009.94.

Hughes, D., Thoelen, K., Horré, W., Matthys, N., Cid, J.D., Michiels, S., Huygens, C. and Joosen, W. (2009) 'LooCI: a loosely-coupled component infrastructure for networked embedded systems', *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, pp.195–203, DOI: 10.1145/1821748.1821787.

ITU (2005) *ITU Internet Reports 2005: The Internet of Things*, International Telecommunication Union (ITU), Geneva.

Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S. and Terziyan, V.Y. (2008) 'Smart semantic middleware for the internet of things', *5th International Conference on Informatics in Control, Automation and Robotics*, pp.169–178.

Kim, J.H., Kwon, H., Kim, D.H., Kwak, H.Y. and Lee, S.J. (2008) 'Building a service-oriented ontology for wireless sensor networks', *Seventh IEEE/ACIS International Conference on Computer and Information Science*, pp.649–654, DOI: 10.1109/ICIS.2008.100.

Lee, J.W., Schulzrinne, H., Kellerer, W. and Despotovic, Z. (2007) 'z2z: 'discovering Zeroconf services beyond local link', *Globecom Workshops*, pp.1–7, DOI: 10.1109/ GLOCOMW.2007.4437805.

Mayer, S., Guinard, D. and Trifa, V. (2012) 'Searching in a web-based infrastructure for smart things', Paper presented at the *Internet of Things (IOT)', 3rd International Conference on the In Internet of Things (IOT)*, pp.119–126, DOI: 10.1109/IOT.2012.6402313, IEEE.

Mealling, M. (2004) *EPCglobal Object Name Service (ONS) 1.0*, EPCglobal Working Draft.

Mulligan, G. (2007) 'The 6LoWPAN architecture', *Proceedings of the 4th Workshop on Embedded Networked Sensors*, pp.78–82, DOI: 10.1145/1278972.1278992, ACM.

Ostermaier, B., Römer, K., Mattern, F., Fahrmair, M. and Kellerer, W. (2010) 'A real-time search engine for the web of things', *The Internet of Things (IOT)*, pp.1–8, DOI: 10.1109/ IOT.2010.5678450.

Petritsch, H. (2005) *Service-Oriented Architecture (SOA) vs. Component Based Architecture*, University of Technology, Vienna.

Rodríguez-Domínguez, C., Benghazi, K., Noguera, M., Garrido, J.L., Rodríguez, M.L. and Ruiz-López, T. (2012) 'A communication model to integrate the request-response and the publish-subscribe paradigms into ubiquitous systems', *Sensors*, Vol. 12, No. 6, pp.7648–7668, doi: 10.3390/s120607648.

Saint-Exupery, A. (2009) *Internet of Things, Strategic Research Roadmap*, *I*nternet of Things Initiative, Surrey.

Santos, L.O.B.D.S., Guizzardi, G., Guizzardi, R.S.S., Da Silva, E.G., Pires, L.F. and Van Sinderen, M. (2009) 'GSO: designing a well-founded service ontology to support dynamic service discovery and composition', *The Enterprise Distributed Object Computing Conference Workshops*, DOI: 10.1109/EDOCW.2009.5332016.

Savidis, A. and Stephanidis, C. (2005) 'Distributed interface bits: dynamic dialogue composition from ambient computing resources', *Personal and Ubiquitous Computing*, Vol. 9, No. 3, pp.142–168, DOI: 10.1007/s00779-004-0327-2.

Sousa, J.P. and Garlan, D. (2002) 'Aura: an architectural framework for user mobility in ubiquitous computing environments', *Software Architecture*, pp.29–43, DOI: 10.1007/978-0-387-35607-5_2, Springer.

Stirbu, V. (2008) 'Towards a restful plug and play experience in the web of things', *IEEE International Conference on the Semantic Computing*, pp.512–517, DOI: 10.1109/ICSC.2008.51.

Terziyan, V., Kaykova, O. and Zhovtobryukh, D. (2010) 'Ubiroad: semantic middleware for context-aware smart road environments', *Fifth International Conference on the Internet and Web Applications and Services (ICIW)*, pp.295–302, DOI: 10.1109/ICIW.2010.50.

W3C (2011) *Semantic Sensor Network XG Final Report*, W3C Incubator Group Report, p.28.

Walewski, J.W. (2011) *Initial Architectural Reference Model for IoT*, EC FP7 IoT-A (257521) D.

Wang, W., Lee, K. and Murray, D. (2013) 'Building a generic architecture for the internet of things', *IEEE Eighth International Conference on the Intelligent Sensors, Sensor Networks and Information Processing*, pp.333–338, DOI: 10.1109/ISSNIP.2013.6529812.

Wei, Q. and Jin, Z. (2012) 'Service discovery for internet of things: a context-awareness perspective', *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*, p.25, ACM, DOI: 10.1145/2430475.2430500.

Xively (2015) *Xively – Public Cloud for the Internet of Things* [online] https://xively.com/ (accessed 16 December 2015).

Yang, S., Xu, Y. and He, Q. (2011) 'Ontology based service discovery method for internet of things', *The Internet of Things (iThings/CPSCom), 4th International Conference on Cyber, Physical and Social Computing*, DOI:10.1109/iThings/CPSCom.2011.104.

Zhang, D., Yang, L.T. and Huang, H. (2011) 'Searching in internet of things: vision and challenges', *IEEE 9th International Symposium on the Parallel and Distributed Processing with Applications (ISPA)*, pp.201–206, DOI:10.1109/ISPA.2011.53. 201.

Zhu, F., Mutka, M.W. and Ni, L.M. (2005) 'Service discovery in pervasive computing environments', *IEEE Pervasive Computing*, Vol. 4, No. 4, pp.81–90, DOI: 10.1109/MPRV.2005.87.

ZigBeeAlliance (2015) *The ZigBee Alliance Creates IoT Standards That Help Control Your World* [online] http://www.zigbee.org/zigbeealliance/ (accessed 12 December 2015).