Garbage Collection of Virtual Object in IoT

Ze-Yi Zhao, Dong Wang
School of Software and Engineering, Shanghai Jiao Tong University, Shanghai, China
E-mail: zeyi.zhao@gmail.com, wangdong@cs.sjtu.edu.cn

***Abstract*-Since the Internet of Things has become more and more popular, the number of devices in IoT increase at a rapid clip. Some Researches introduced virtual object to enrich the management of the devices. Therefore, virtual object life cycles will need to be thoroughly managed, so that when their usage changes or they are not needed anymore, the virtual objects should be updated and deleted correspondingly. However, there is no specific way in IoT to execute garbage collection of virtual objects. This paper proposes a garbage collection method for virtual objects that uses the concept smart device to manage the virtual objects' life cycles and solve the circular references problem when using the smart device.**

Keywords-virtual object; garbage collection; smart device; circular references

I. INTRODUCTION

In IoT apps, current interaction model has shifted from based on humans looking for information provided by objects (human-object interaction) to the object-object interaction. The object-object interaction is based on objects looking for others to provide composite services, which increases the interaction complexity. To cope with this challenge, the physical devices' capabilities need to be augmented and devices should be able to talk to each other at the same level to make the realization of robust applications easier. The virtualization of objects is the perfect answer to this problem.

Major IoT platforms have introduced their own vision of the virtual objects. The role of the virtual objects is to bridge the gap between the physical and the virtual worlds. To achieve this goal, the virtual object is intended to support:

- Fast deployment of new services related to the physical world;
- Co-existence of heterogeneous objects over a common infrastructure;
- All-time reachability of real objects;
- Self-management of network objects through context awareness

Virtual objects need to be described semantically in order to expedite the discovery of services they provide and to make heterogeneous objects interoperable at the virtual layer. Based on the characteristics and functionalities, virtual object can be defined as followed: a virtual object is a digital representation of a real world object, which is able to acquire and analyze the information about its context, and to augment the service experience of the associated devices.

More and more researches on device virtualization have been carried out, and some problems are gradually exposed with the introduction of device virtualization. Interoperability is one of the major problems, although there are some virtualization-based IoT solutions that have similar functionalities, there are neither standard formats nor recommendations to regulate the virtual objects' usage. This leads to the situation that virtual objects belonging to different architectures can only communicate with each other in some cases, which makes the cooperation between these virtual objects more and more difficult.

On the other hand, with the upsurge of the objects in the IoT, the scalability issue should also be concerned. Virtual object life cycles need to be thoroughly managed, when some objects' usage changes or they are no longer needed, they should be updated and deleted correspondingly. Especially when some virtual object is shared among multiple IoT apps and this virtual object is going to be deleted, the robust of the app should also be considered, crash will not be allowed because of one virtual object's deletion.

This paper is mainly about a new garbage collection method targeting virtual objects. Firstly, the method for the management of virtual objects life cycle is proposed. Secondly, based on reference counting method used in java garbage collection system, we propose a new garbage collection method, to solve the circular references problem in reference counting, we introduce a concept smart device, which is a semantical enrichment of the original virtual object.

The rest of this paper is organized as follow: section 2 presents related works. Section 3 describe the details the garbage collection method. Implementation is given in section 4 and discussion of the results and concluding remarks end this paper in section 5 and section 6.

II. RELATED WORK

Current researches about the management of virtual objects includes the implementation, management and the application. [6]presents the evolution of its definitions, current functionalities assigned to the virtual object and how they tackle the main IoT challenges, and major IoT platforms. The first attempts towards the virtualization of real world objects were connected with Radio Frequency Identifiers (RFID) [7] which could only capture raw data. Next steps were done in the contextualization of captured data [8]. And then, the emergence of virtual objects bridge the gap between physical word and virtual world.

FI-WARE [9] is based upon Generic Enablers (GEs), which offers reusable and commonly shared functions, serving a multiplicity of usage areas across various sectors. In FI-WARE, sensors are modelled as virtual objects that have an ID, a type and several attributes, so that every object is associated to one single virtual object. SENSI [10] enables the integration of heterogeneous and distributed Sensor and Actuator Network (SAN) islands into a homogenous framework for real world information and interactions. It provides an abstraction level of resources corresponding to the real world consisting of Real World Entities. SENSI providing a many-to-one association between real and virtual objects. IoT-A [11] extends the models introduced in SENSEI and proposes an architectural reference model for the IoT. IoT-A introduces entity, which is the core of the project. Entities are associated with resources, which can be accessed through the interfaces provided for users. One physical object is corresponding to many virtual entities. iCore [12] is organized in three levels: the Virtual Object(VO) level, VO is the virtual representation of real world objects that are dynamically created and destroyed. In the upper layer, the Composite Virtual Object (CVO) level, CVO is a mashup of several VOs defined by some specific functionality or service request. The last layer is the Service layer, it translates the apps' requirement into the service that CVO need.

[1][3] adds cognitive management to iCore [12], in VO level, there is a constant link between physical devices and virtual objects, through which the device may complete its self-management and self-configuration. Cognitive management makes virtual objects more intelligent.

Currently, the garbage collection towards virtual objects is not proposed. This paper will introduce a garbage collection method towards virtual objects based on java garbage collection methods. [5] presents some garbage collection methods used in java virtual machine. Normal methods includes reference counting, mark-sweep, copying, mark-compact and generational collecting. The garbage collection method we proposed is based on the reference counting method. The main idea of reference counting is to use reference counter, for an object A, if another object references A, the reference counter of A increases, when the reference become invalid, the reference counter decreases, when the counter reaches 0, then A will not be used any more.

III. GARBAGE COLLECTION METHOD OVERVIEW

A. Virtual Object Life Cycle Management

Virtual objects bring convenience to IoT. In the meantime, the management of virtual objects should also be concerned. The whole process from the construction to the destruction of the virtual objects should be managed, there is some constant link between physical device and virtual objects, which is maintained by the heartbeat server. The heartbeat server pings the virtual object once in a while to ensure its connectivity. Though this link, we can implement the management of physical devices by virtual objects.

This paper is about the extension of cognitive management system. In normal cognitive management system, VO and CVO makes the many-to-many map between physical devices and virtual objects possible. It makes great use of physical devices by enabling application to use these physical devices while these devices belong to different context. All virtual objects are registered in the registry, and when the life cycle of the virtual object is over, garbage collection is executed.

B. Garbage Collection Mechanism

Garbage collection is executed when some virtual object's life cycle is over. This paper proposed a new garbage collection mechanism:

(1) When a physical device is out of its context, the garbage collector will look up in the VO registry and the CVO registry to find all records related to the device, and execute garbage collection on this device.

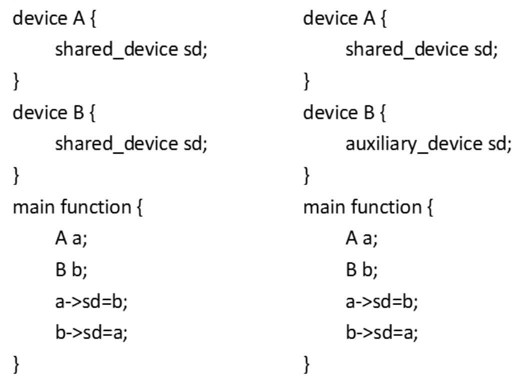
(2) When some IoT application terminates and some virtual object's life cycle is over, the garbage collector will look up in the VO registry and the CVO registry to find all records related to the device. If not found, then garbage collection is executed, otherwise, decrease the reference counter of the virtual object.

(3) When it involves both VO and CVO, we need to consider that CVO is composed of VOs, all references to CVO will influence the references to VO. When multiple IoT applications use the same CVO, obj. when obj's reference counter changes, the reference counters of the VOs that consists obj will not change. VO's reference counter only increases when another CVO references to it. When CVO is destructed, the reference counter of the VOs that consists obj will decrease accordingly.

There is one big flaw in java reference counting, which is reference-counting. When two virtual objects references each other, then both of these virtual objects' reference counter cannot reach 0, which leads to the storage space waste. To this, this paper designed smart device, it's an extension of virtual object and it solves the reference counting problem.

C. Smart Device

Smart device is divided into shared device and auxiliary device. Shared devices provide services that are used to control the real world objects. Shared devices also maintain the reference counters of the virtual objects, and auxiliary devices don't keep the reference counters, they are used to monitor the usage of shared devices. When circular references are required between virtual objects, we may use auxiliary devices to avoid the circular references. As showed in Fig. 1:



```
device A {                        device A {
      shared_device sd;                 shared_device sd;
}                                 }
device B {                        device B {
      shared_device sd;                 auxiliary_device sd;
}                                 }
main function {                   main function {
      A a;                              A a;
      B b;                              B b;
      a->sd=b;                          a->sd=b;
      b->sd=a;                          b->sd=a;
}                                 }
```

Figure 1. Cases before and after using auxiliary devices

Fig. 1-a shows codes before we use auxiliary device, and Fig.1-b shows codes after using auxiliary device. Auxiliary device itself doesn't increase the reference counter of the virtual object, so no circular reference problem will occur.

This paper added manager object to smart device to implement the sharing and counting, manager object is the encapsulation of managed device. Reference counters used in shared devices and auxiliary devices actually counts the references to this manager object. Fig. 2 shows how the smart object works

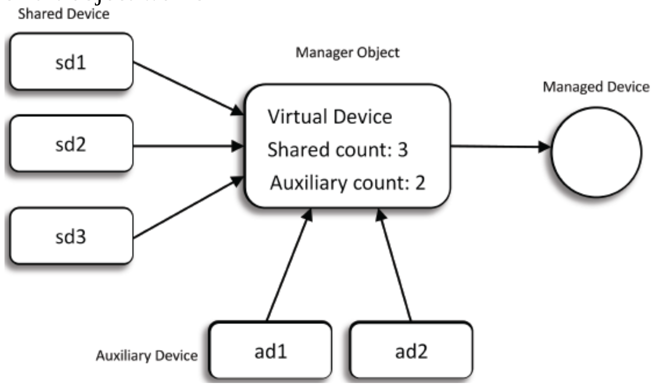


Figure 2. How smart device works

The process begins when the managed object is dynamically created and the first shared device sd1 is created to point to it; the constructor of shared device dynamically creates a manager object, which contains the real managed device. The manager object also contains two reference counters: the shared count counts the number of shared devices pointing to the manager object, and the auxiliary count counts the number of auxiliary devices pointing to the manager object. When sd1 and the manager object are firstly created, shared count will be 1 and the auxiliary count should be 0.

If another shared device sd2 is created with sd1, it also points to the same manager object and the constructor increments the shared count, and now there are 2 shared devices pointing to the managed object. Likewise, when an auxiliary device is created with a shared device or another auxiliary device, they points to the same manager object and the auxiliary device is incremented. Fig. 2 shows the situation after 3 shared devices and 2 auxiliary devices have been created to point to the same object.

When a shared device is destroyed or reassigned to point to another object, the shared device destructor decrements the shared count. Similarly, destroying or reassigning an auxiliary device will decrement the auxiliary count. When the shared count reaches 0, the manager object will be deleted and the managed device will be deleted from the registry, but if the auxiliary count is greater than 0, the manager object is kept. If the auxiliary count reaches 0 too, the manager object will be deleted. In general, the managed device stays as long as there are shared devices pointing to it, and the manager object stays as long as there are either shared devices or auxiliary devices referring to it.

Shared device and auxiliary device have a basic difference: shared device can be used almost identically to a virtual object, in fact, it will provide all the interfaces that the virtual object can provide. However, an auxiliary device is much more limited. Users cannot use it like a virtual object, more specifically, it cannot be used to actually refer to the managed device at all. The only operation that an auxiliary device can provide is to interrogate it to see if the managed device is still there. If the managed device is gone, the shared device will be an empty one; if the managed device is present, then the shared device can be used normally.

IV. IMPLEMENTATION

This paper is based on intel-iot-services-orchestration-layer[13], which is a total solution that provides visual graphical programming for developing IoT applications. We modified the entity-store part of the project and added smart device and garbage collection method.

A. Smart Device

Smart devices are divided into shared devices and auxiliary devices. When a real world object registers in the IoT system, a virtual object is created dynamically. The owner of the device can set the priority level of the device, and the device manager will create smart devices according to the priority level. The interfaces that the shared devices and auxiliary devices provide are shown in Tab. 1 and Tab. 2.

TABLE I. INTERFACES OF SHARED DEVICES

Construct(managed_device md, priority)	create shared device with device and its priority
Construct(shared_device sd)	create shared device with another shared device
Reset	reset shared device to make it not point to the original device
RefCount	get the reference count to the shared device
GetAuthority	get the priority level of the shared device
Destruct	destroy shared device

TABLE II. INTERFACES OF AUXILIARY DEVICES

Construct(shared_device sd)	destroy shared device
Construct(auxiliary_device ad)	create auxiliary device with another auxiliary device
Reset	reset auxiliary device to make it not point to the manager object
RefCount	get the reference count to the shared device
GetShared	get the shared device
Destruct	destroy auxiliary device

B. VO & CVO Registry

VO is the virtual map to the real world object, when a real world object is register, the system creates VO according to the context and the metadata of the device. VO includes status, owner, etc. VO is stored in the system as a key-value pair, every VO consists of device ID and device object, and VOs are stored in the VO registry which is stored in a specific file in the file system. CVO is comprised of several VOs and is also stored in the CVO registry. When receiving the request to create composite service from upper level, the management unit searches the CVO registry for an existing CVO that can provide the service. If such a CVO is unavailable, then it perms a look up in the VO management unit to find relevant VOs that belong to the CVO. The CVO holds the meta-data of the VO, along with details to connect to the VO and real world object in turn.

C. *Construction & Destruction of Smart Devices*

The construction of smart devices requires the functionalities and priority level the real world object. At first, every smart device is assigned a unique ID as the key of the key-value pair, then a virtual object is created according to the functionalities as the value of the key-value pair.

The destruction is more complicated. For deletion of CVO, the CVO management unit searches the CVO registry to find the CVO and decrement the reference count by 1, if the reference count reaches 0, then CVO is erased from the registry, and the VOs that the CVO contains will update their reference counts. Similarly, for deletion of VO, the VO management unit performs a look up in the VO registry and decrements the reference count by 1, if it reaches 0, then delete the VO from the VO registry.

V. RESULTS AND DISCUSSION

With the fully functional implementation, tests were carries out to understand the timing information for the garbage collection of the smart devices. Simulations were performed varying the number of VOs in the VO registry. It has been observed, based on the current implementation, the garbage collection method is correct and the time taken to execute garbage collection didn't show a great increase. We tested in 3 different applications in order to ensure the correctness of the results and as shown in figure, our garbage collection method shows good performance as the number of VO increases.

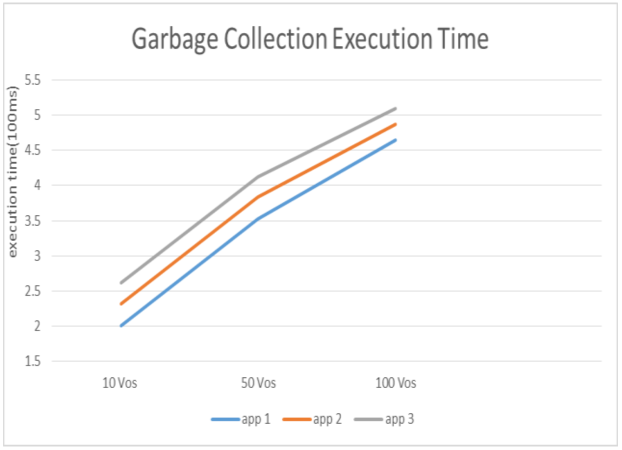


Figure 3. Execute time of garbage collection.

VI. CONCLUSION

Virtual objects play an important role in IoT applications, and how to manage virtual objects efficiently and properly is a severe problem. Garbage collection towards virtual objects has not been mentioned in current virtual object management systems. This paper proposes a garbage collection method based on reference counting method. In order to settle the circular references problem, we introduce smart device.

In future works, we will concentrate on the usage of the smart device, it is only used for garbage collection for now. Priority level of virtual objects and management of users information will also be considered.

REFERENCES

[1] D. Kelaidonis, A. Somov, V. Foteinos, G. Poulios, V. Stavroulaki, P. Vlacheas, P. Demestichas, A. Baranov, A. R. Biswas, and R. Giaffreda, "Virtualization and cognitive management of real world objects in the internet of things," in Green Computing and Communications (GreenCom), 2012 IEEE International Conference on. IEEE, 2012, pp. 187–194

[2] Vlacheas P, Giaffreda R, Stavroulaki V, et al. Enabling smart cities through a cognitive management framework for the internet of things[J]. IEEE communications magazine, 2013, 51(6): 102-111.

[3] Foteinos V, Kelaidonis D, Poulios G, et al. Cognitive management for the internet of things: a framework for enabling autonomous applications[J]. IEEE Vehicular Technology Magazine, 2013, 8(4): 90-99.

[4] Foteinos V, Kelaidonis D, Poulios G, et al. A cognitive management framework for empowering the internet of things[C]//The Future Internet Assembly. Springer Berlin Heidelberg, 2013: 187-199.

[5] Venners B. Inside the Java virtual machine[M]. McGraw-Hill, Inc., 1996.

[6] Nitti M, Pilloni V, Colistra G, et al. The Virtual Object as a Major Element of the Internet of Things: a Survey[J]. IEEE Communications Surveys & Tutorials, 2015, 18(2): 1228-1240.

[7] R. Weinstein, "RFID: a technical overview and its application to the enterprise," IT Professional , vol.7, no.3, May-June 2005, pp. 27- 33. [Online]. Available: IEEE Xplore Digital Library, doi: 10.1109/MITP.2005.69. [Accessed: Oct. 2012].

[8] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for Data Processing in Large-Scale Interconnected Sensor Networks", Proc. of the International Conference on Mobile Data Management (MDM 07). Washington, DC, USA, May 2007, pp. 198-205. [Online]. Available: ACM Digital Library, doi: 10.1109/MDM.2007.36. [Accessed: Oct. 2012].

[9] FI-WARE. (2011–2014). Core platform of the Future Internet [Online]. Available: http://www.fi-ware.org/, European Commission under the 7th Framework Programme

[10] SENSEI. (2008). Integrating the Physical With the Digital World of the Network of the Future [Online]. Available:

[11] IoT-A. (2010). Internet of Things—Architecture [Online]. Available: http://www.iot-a.eu/

[12] iCore. (2011). Empowering IoT Through Cognitive Technologies [Online]. Available: http://www.iot-icore.eu/

[13] intel-iot-services-orchestration-layer(2015). The total solution that provides visual graphical programming for developing IoT applications[Online]. Available: https://github.com/01org/intel-iot-services-orchestration-layer