Research Article

IoTSilo: The Agent Service Platform Supporting Dynamic Behavior Assembly for Resolving the Heterogeneity of IoT

Euihyun Jung,1 Ilkwon Cho,2 and Sun Moo Kang2

1 Department of Computer Science, Anyang University, 602-14 Jungang-ro, Buleun-myeon, Ganghwa-gun, Inchon 417-833, Republic of Korea
2 Division of Digital Infrastructure, National Information Society Agency, NIA Building, Cheonggyecheonno 14, Jung-gu, Seoul 100-775, Republic of Korea

Correspondence should be addressed to Ilkwon Cho; ikcho@nia.or.kr

Received 30 August 2013; Accepted 28 November 2013; Published 14 January 2014

Copyright © 2014 Euihyun Jung et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Although a lot of researchers have painted a rosy picture of Internet of Things (IoT), there have been no widely accepted solution and related standards until now. To achieve the successful realization of IoT, the close collaboration of devices is the primary requisite. However, the heterogeneity of devices such as different hardware or network connectivity prohibits the realization of IoT. In order to overcome the heterogeneity issue, we suggested the agent service platform named ioTSilo in which agents can communicate and cooperate on behalf of their devices. With this delegation approach, the ioTSilo can support diverse devices without worrying about their differences. In designing an agent, several software design patterns are adopted to enable the agent to assemble behaviors for hiding the heterogeneity of devices. To investigate the effectiveness of the ioTSilo, we developed eleven different types of the IoT devices to emulate real world things with Arduino, deployed the devices in both Korea and Japan, and then conducted three experiments.

1. Introduction

A lot of research groups have predicted that Internet of Things (IoT) would be a core infrastructure to provide the future IT services enabling the smart society [1–3]. IoT is defined as a pervasive network infrastructure where various digital things including RFID embedded objects, sensors, or mobile phones are interconnected and communicate with each other. In the near future, several trillion devices would collaborate to serve people through the global machine-to-machine interaction provided by the IoT technology [4–6]. However, the research on IoT is still in its early stage, so there is no agreed and common solution to realize it yet [7–9].

There are several issues to block the IoT realization; the most essential one is the heterogeneity caused by the various features of devices such as network types, identifier schemes, and interworking protocols [3, 4]. IoT needs a close collaboration of devices, but it is hard for the devices with the different features to interact with each other. For example, wireless sensors and RFID embedded devices cannot identify peers due to their different identifier schemes. Even, the same kinds of wireless sensors could not make a session if they have different network connectivities. Unfortunately, the current IoT enabling technologies such as wireless sensor networks (WSN) [10] and machine-to-machine (M2M) [11] cannot cope with these problems properly yet. Although WSN is considered as the main infrastructure of IoT [6], WSN usually consists of homogeneous devices sharing the same network types and protocols. As a result, WSN often makes isolated islands that cannot interoperate with each other, so it cannot work as the IoT technology by itself anymore. M2M has existed in the different form since the advent of computer networking automation such as “telemetry” or “SCADA.” Due to the stable cellular M2M communication and the sophisticated M2M data modules, M2M is considered as another prominent technology for IoT. However, M2M is driven under the telco business model, so it is inherently
locked in the walled garden of telcos. That is, M2M also makes a closed island similar to WSN except its size. In addition to this, M2M companies forced to use their proprietary M2M module to join their M2M networks, and thereby the third party groups had difficulty in joining it. For now, both WSN and M2M do not propose any particular ways to resolve the heterogeneity, whereas they stick to the monolithic design of the same hardware, network types, and even protocols [4].

In order to resolve the heterogeneity issue of IoT, the researchers of cyber-physical systems (CPS) have paid attention to the software agent and they believe that the agent technology would be a solution to hide nicely the heterogeneity of devices [12, 13]. In the proposed scenario, devices delegate their authority to the corresponding agents and then the actual collaboration is done in the agent level [12]. Since an agent identifies and communicates with other agents in the agent space, the differences of physical devices can be easily hidden. This seemed a good idea to use the agent technology for overcoming the heterogeneity, but a detailed structure or a reference model was not represented yet. Most of all, to hide the heterogeneity, the single type of agent has to contain all different functions of heterogeneous devices in it. It can be an intuitive approach, but it is impractical because all features and functions of various devices cannot be anticipated and implemented in the development phase. Additionally, even if this approach is adopted, whenever a new device appears or the functions of a device are changed, the agent must be modified in the source code and be redeployed again into the system. Therefore, new approach with which an agent transforms itself depending on the features of devices is needed.

In order to fulfill the requirement, we proposed an agent architecture in which an agent assembles the needed behaviors for the corresponding device and links them into its core on runtime like the Lego block. Since this architecture is much different from the ordinary software design, we adopted portable service abstraction (PSA) and dependency Injection (DI) design patterns. We also developed the agent service platform named iotSilo to evaluate the proposed agent architecture. In the iotSilo, various devices with different sensing functions and network connectivities delegate their authority to the corresponding agents. We conducted several experiments on the iotSilo after making various devices with Arduino [14] and deploying the devices in both Korea and Japan. The experiments showed that the proposed agent can assemble various behaviors depending on the corresponding devices and it enables acting for the totally different devices without affecting the agent itself at all. They also demonstrated that various devices with different network connectivities can collaborate through the cooperation of their agents instead of direct communication between heterogeneous devices. This evaluation indicated that the proposed agent model can be a novel method to overcome the heterogeneity of IoT.

The remainder of this paper is organized as follows. The related research about the heterogeneity of IoT is stated in Section 2. In Section 3, the proposed platform is minutely described. This is followed by Section 4 which discusses several experiments to show the effectiveness of the proposed platform. Finally, Section 5 gives the conclusion.

2. Related Research

There are several fundamental causes why existing technologies have failed to resolve the heterogeneity issue of IoT. First, devices have different identification schemes such as IPv6, RFID, or ZigBee address, so they cannot identify and locate others. In the conventional data network, a participant has to identify and locate peers using identifiers before it makes a session. IPSO [15], Auto-ID [16], and OID [17] research groups have been competing for occupying the major identifier in IoT, but any group does not answer the interoperability with other identifiers. Second, the different network connectivity inhibits devices from making an end-to-end session between them. For example, a device on a ZigBee network cannot make a session to another device on the Internet with IP connection. There have been some studies to address this [18, 19], but they partially solved the problem using a specialized gateway. Lastly, there is still no agreed common interworking protocol between heterogeneous devices. Although Jini, UPnP, HAVi, and so forth were suggested as the interworking protocols, they all failed to unite devices or to interoperate with other standards. Since devices are made from many different manufacturers and they equip their own interworking protocols, they cannot exchange information at all even if devices manage to make a physical session between them.

While existing technologies failed to show their abilities for the IoT realization, some research groups suggested the alternative architecture with a totally different approach. This architecture is called the IoT data platform [20]. Several platforms such as COSM (formerly Pachube) [21] or Open.sen.se [22] are announced. In the IoT data platform, the IoT devices just have a responsibility to report their data using a predefined protocol while they ignore their heterogeneity such as the network types. The reported data can be used to provide the instant IoT services to its users regardless of the kinds of the reporting devices. The IoT data platform achieves some degree of success because it can hide the heterogeneity by separating devices and their data. COSM reports one million data feed into COSM per day, and the developers of COSM can easily make new IoT services with the reported data. However, this approach cannot provide a peer-to-peer collaboration between devices, because its purpose is to manipulate the transmitted data instead of accessing the devices directly. Separating devices and their data is a good attempt to make mash-up IoT applications, but the full-level IoT services such as the device collaboration cannot be achieved with this architecture.

Besides, the CPS researchers expected that the software agent played an important role to realize IoT [12, 13]. It seems a novel approach, but they have not provided a reference model or a practical architecture until now. The most controversial issue of this approach is how an agent can hold vast functions of heterogeneous devices within it. It is impractical that the single agent contains every function of all
devices in the world. The intuitive solution for this issue can be the creation of all kinds of agents for devices. However, this solution also has a limitation. Different kinds of agents can suffer with the interoperability between them as their corresponding devices do.

3. Architecture

3.1. Conceptual Design. The heterogeneity of devices is natural and unavoidable because devices have to be designed to fit themselves to their applications and surroundings. Therefore, the forcing uniform scheme on the heterogeneous devices has no choice to fail. We decided that hiding the heterogeneity would be a key design factor instead of enforcing particular network types, identifier schemes, or protocols. For this purpose, we adopted a software agent [23] to act for a device in a relationship of agency. An agent can communicate with other agents while it hides its device's network type, protocol, and other heterogeneous natures. Even while devices are located on the different networks and have different identifiers, they can painlessly communicate and collaborate with each other in the agent space. This approach enables the physical interaction of devices to be transposed onto the cyber interaction of agents, which is greatly flexible in responding to the heterogeneity of devices. Comparing to the severe limitation of the physical environment, the cyber environment allows devices a high level of freedom to collaborate with each other regardless of their different natures.

The conceptual design of the iotSilo adopting the software agent is shown in Figure 1. In the iotSilo, a device delegates its authority to a corresponding agent. When a device wants to communicate or collaborate with other devices, it orders its corresponding agent to do it. Then, the corresponding agent performs a proper behavior on behalf of the device, such as reporting the device's data to the system or sending a request to other devices. If needed, the agent can communicate with other agents using the message bus in the iotSilo. Using the iotSilo, devices can make a logical channel between them using only their limited physical channels connected to their agents even if they cannot directly communicate with each other.

3.2. Agent Design. In designing agents, the extensibility is a major consideration that we have thought intensely, because an agent should handle many various devices without corrupting the running system. Generally, there are two main methods of the extensibility in the object oriented design (OOD): Inheritance and Composition. Although Inheritance is intuitive, the method is not recommended to make different kinds of classes inherited from the same parent class [24]. Inheritance should be used in the “is-a” relationship between classes. That means all classes using Inheritance are included in the same category. In the research, agents seem to be the same kind of “agent” which acts for a real world thing. However, agents are just the avatar of real world things which are totally different in their properties and operations. Therefore, the adoption of Inheritance is not proper for this case. Also, we were reluctant to choose Inheritance because of the several weaknesses: inheritance conflict, excessive dependency of intermediate parent classes, and compile time binding [24].

Comparing to Inheritance, Composition provides more flexibility. By dividing an agent into agent’s core and its behaviors which perform the functions of its device, the agent can easily accept any kind of request from devices. However, the coherence between the agent core and its behaviors is still unavoidable even though Composition is adopted. In the research, the portable service abstraction (PSA) and dependency injection (DI) patterns are used to remove the coherence.

3.3. Portable Service Abstraction (PSA). In order to support every function of the various devices, the separation of the functions from the agent core is required. However, this separation leads to a new problem that an agent core has to know all method signatures of the device-dependent functions. That is, the agent core has no choice but to contain the name of all classes and method signatures in its source code as shown in Figure 2(a). In this structure, whenever a new device is added, the agent core should be modified in the source code level and be redeployed. Therefore, a simple separation cannot be a solution in considering the vast diversity of devices.

This problem has been referred to as the dependency in the OOD and the portable service abstraction (PSA) policy has been proposed to reduce the dependency between objects. The PSA recommends architects to use an interface rather than a concrete object for achieving low coherence between objects. In our design shown in Figure 2(b), devices’ functions are abstracted as a Behavior interface, which is implemented by actual behavior classes. Then, the agent core can call various device-dependent functions through the Behavior interface. Unlike the conventional system, the addition of new devices can be easily supported without affecting the agent core and the running system. The resulting structure is to the form of the Strategy design pattern. With the pattern, the agent core can act as a proxy of various devices by equipping various device-dependent classes which implement the Behavior interface.

3.4. Dependency Injection (DI). The PSA policy provides an agent to equip the behavior classes performing device-dependent functions without modifying the agent core, but
the agent core still has to hold the behavior classes’ names in the source code because the instantiation of the behavior classes is unavoidable. We resolved this problem using the dependency injection (DI) [25]. The DI is a design pattern that allows removing hard-coded dependencies in the source code and injecting them from the external configuration. The DI ensures that the agent core does not know where device-dependent functions come from and it allows the selection among multiple implementations of the given interface at runtime. Using the PSA and the DI, the agent core can choose the proper behavior class from the external configuration without knowing the actual name of the behavior class as shown in Figure 2(b). In the snippet of the configuration in Algorithm 1, the “vibAgent” acts for a vibration sensor and it has three directives: a “report,” a “get_data,” and a “set_cmd.” Generally, a directive indicates an agent’s behavior open to the external entities such as other agents. The external entities can request a behavior to the target agent with the directive corresponding to the behavior instead of the actual class name of the behavior. With this mechanism, the same directive can be mapped to different behavior classes depending on the type of devices. The directives are linked to the actual implemented classes in the configuration file shown in Algorithm 1 and these classes are dynamically assembled into the agent core at runtime. For example, the “report” behavior is linked to the “GeneralReport” class that reports the vibration sensor’s data to the iotSilO management console.

In the proposed system, all the behavior classes should implement the Behavior interface which has a single method of “process (object... values)” and they are assembled into agents using the dynamic binding. In general, the Strategy pattern is enough to change behaviors depending on the types of devices. However, an agent has to support many functions of devices, so the agent maintains a behavior map which holds the pair of the “directive” key and the “Behavior implemented class object” value. In Figure 3, the vibAgent’s behavior map based on the configuration file in Algorithm 1 is described. When an external directive “report” is requested to the vibAgent, the agent chooses the corresponding “GeneralReport” behavior class and then

---

**Algorithm 1:** The snippet of configuration file describing the vibAgent and its behavior classes.

```xml
<bean id="vibAgent"
    class="net.agentsilo.iotsilo.agent.ThingAgent" scope="prototype">
    <property name="info" ref="thingInfo"/>
    <property name="behaviorMap">
        <map>
            <entry key="report" value="vib_report"/>
            <entry key="get_data" value="vib_get"/>
            <entry key="set_cmd" value="vib_set"/>
        </map>
    </property>
</bean>

<bean id="vib_report"
    class="net.agentsilo.iotsilo.behavior.GeneralReport"/>
<bean id="vib_get"
    class="net.agentsilo.iotsilo.behavior.DefaultHttpGetData"/>
<bean id="vib_set"
    class="net.agentsilo.iotsilo.behavior.DefaultHttpSetCmd"/>
```

---

**Figure 2:** (a) The ordinary dependency. (b) The decreasing dependency by using the Behavior interface.
executes it. Using the method, agents can hold any kinds of devices’ behaviors unlimitedly.

3.5. Protocols. In the iotSilo, two kinds of protocols are defined. First protocol is between a device and a corresponding agent and second is between agents. A protocol between a device and an agent should support exchanging various data with devices because devices can have their own unique properties and operations. However, if all requirements are defined in a single protocol interface, the protocol interface would be too complex. It means that the protocol should be not only extensible but also concise.

In order to meet both conflicting requirements, we designed the protocol interface consisting of a common and a custom part as shown in Figure 4. In the common part, “version” is the protocol version and “id” is the identifier allocated to a device and its agent. The common part is not unchanged and all agents handle it identically, but the processing of the custom part tagged as “data” can be varied depending on the devices’ nature. When a packet is received from its device, an agent processes the common part and it hands over the custom part to its corresponding behaviors using the given directive defined in the custom part. The “arg” in the custom part can be varied because the handling method of the Behavior interface supports variable arguments. That is, agents leave the manipulation of the custom part to the indicated behavior, so the contents of the custom part can have a high degree of freedom.

The protocol interface in Figure 4 is used in the communication flow from a device to an agent. In order to collaborate with other devices, the protocol should support the bidirectional communication, so it should have an interface called from an agent. In the opposite flow, the interface format is the same except for the number of directives. We decided that the protocol interface toward the device can have only a single directive in the custom part considering the processing power of an embedded device.

The protocol interface between agents is similar to the one between a device and an agent as shown in Figure 5. The different part is that the protocol interface can only hold a single behavior. A device can issue multiple behaviors at the same time, but an agent only requests a single behavior at one time. The protocol interface between agents can also contain any kind of information in the custom part because the corresponding agent leaves actual processing of the custom part to the matching behavior. The “id” is a sender identifier and a receiver identifier is omitted because each agent has its own listening channel to the message bus in the iotSilo. When a sender agent publishes a message to the target channel of the receiver agent, the message is asynchronously delivered through the message bus.

3.6. Message Bus between Agents. Between agents, multiple concurrent communication sessions could occur in the iotSilo. Since all sessions are independent of each other, the ordinary request-response metaphor, for example, synchronous communication between agents, may cause the performance degradation and the deadlock of the system. Small amounts of mesh topology connections among agents are enough to halt the whole system if only a single agent delays a response. In order to avoid this catastrophe, we adopted a message oriented middleware (MOM) [26] to construct a message bus.
MOM relies on asynchronous message passing in which a message queue provides temporary storage when a destination peer is busy or is not connected. Therefore, a sender and a receiver do not need to connect to the network at the same time. It can solve problems with the intermittent connectivity. Even if a receiver fails for some reasons, senders can continue to send their messages unaffected. The messages will simply accumulate in the message queue for later processing when the receiver resumes. In the iotSilo, an agent is designed to have a listening channel named its identifier on the message bus. When an agent wants to send a message to a peer agent, the agent makes a message and it publishes the message with the peer agent’s identifier. Then, the message bus in the iotSilo delivers the message to the destination agent asynchronously. This process is depicted in Figure 6.

IoT is expected to be composed of at least several million devices and various messages are exchanged in it. Therefore, the extensibility and the scalability are major considerations to choose MOM. In order to fulfill these design factors, we selected ActiveMQ [27] for the following reasons. First, ActiveMQ supports both string messages and blob message. In the current design, the protocol only uses string messages, but if the binary data is needed, the protocol can be easily extended by using the blob message. Second, ActiveMQ has a capability to build a cluster easily. The capability is important when the message bus of the iotSilo is required to be scaled up. The plan for the scalability of the iotSilo will be discussed in Chapter 5.

4. Experiments

4.1. Physical Things and Experiment Environment. In the research, the IoT devices were implemented using Arduino [14] in order to emulate real world things. Arduino is an open-source single-board microcontroller designed to make various electronic devices easily in multidisciplinary projects. We made eleven different types of the IoT devices: a temperature sensor, a humidity sensor, an optical detector, a vibration sensor, a radioactive sensor, a camera sensor, a rainfall meter, a wind vane, a wind gauge, a speaker, and an illuminometer. The devices were connected on three kinds of networks: Ethernet, WiFi, and ZigBee. These totally different functions and network types enable the IoT devices to emulate real world heterogeneous things. The developed IoT devices are shown in Figure 7.

In order to show the effectiveness of the iotSilo, the IoT devices were deployed at three locations: Anyang University and Pusan University in Korea and Waseda University in Japan. The points in Korea were connected to Korea Advanced Research Network (KOREN) and the point in Japan was connected to Japan Gigabit Network Extreme (JGN-X). Each point was configured to have eleven devices.
as shown in Figure 8. In these experiments, we wanted to test three check points: the dynamic assembly of the various device-dependent functions into the agent's core, the collaboration between devices at a distant location with different network types, and the modification of behaviors without affecting other parts of the system.

4.2. Dynamic Assembly of Device-Dependent Functions. In modeling the agent, we focused on the agent's dynamic support of the device-dependent functions without modifying the source code of the agent itself. To evaluate this design purpose, we conducted an experiment in which the same kinds of agents act as a proxy of two different kinds of sensors: a camera sensor connected through the Ethernet and a temperature sensor through the ZigBee.

The camera sensor reports binary image data to the corresponding agent which stores the data into a mass storage and renders the stored images to the management console. On the other hand, the agent of the temperature sensor stores the reported temperature data into a database and it renders the data as a bar chart. As shown in Figure 9, each agent does not need to modify the agent core at all but just to load proper device-dependent behaviors in the external configuration as shown in Algorithm 2. At runtime, each agent loads device-dependent behaviors and assembles them into the agent itself as indicated in the configuration. This experiment showed that any kinds of devices can be easily supported by the proposed agent without the code modification if the corresponding device-dependent functions are indicated in the external configuration.

After the experiment, we changed the report function from the "CelsiusReport" class to the "FahrenheitReport" class in the configuration of the agent for the temperature sensor. This subexperiment assumed the situation where a device changed its behavior after finishing the deployment of the IoT services. In this sub experiment, the reported temperature data is well interpreted as the Fahrenheit value without affecting other parts of the system and even the corresponding agent itself.

4.3. Collaboration of Heterogeneous Devices at Distant Locations. In this experiment, when an optical detector at Waseda University sensed the human body, it sent a request to a speaker at Anyang University to make a warning sound. Physically, the optical detector and the speaker were not directly connected and did not know each other's network status at all, but the optical detector communicated with the speaker using agents' collaboration. The detector issued the "report" directive to its agent; then the agent chose the "humanDetection" behavior class which sent a request message containing "beep" directive to the speaker agent. After receiving the request, the speaker agent chose the behavior corresponding to the "beep" directive. Then, the behavior issued a "set_cmd" directive which contained a sound generation command to the speaker. The experiment gave the expected result although these devices were connected on different networks. The whole procedure is shown in Figure 10.

4.4. Addition of New Functions without Affecting Other Parts of the System. We pointed out that new devices would be
often added and the already deployed devices would replace functions due to the dynamicity of IoT. In the experiment, we assumed that a radioactive sensor suddenly needed a warning sound for the high radioactivity level. However, the radioactive sensor did not have a sound function at the deployment time, so it needed a help from the speaker nearby. To add the function after the device deployment, we just only changed the radioactive sensor’s “report” behavior from the “GeneralReport” behavior to the “RadioActiveReport.” The “RadioActiveReport” behavior sent a request to the speaker agent when it checked the high radioactivity level. In order to check this function, the radioactive beads were used as shown in Figure 11.

The result showed that the addition of new function was successful without affecting other parts of the system. It means that the mash-up-style IoT services are always possible with the iotSilo. Additionally, though the radioactive sensor was connected with WiFi and the speaker was connected with
ZigBee, two devices easily communicated with each other in spite of the difference of their network connectivity.

5. Evaluation and Future Research

5.1. Evaluation. In order to evaluate the effectiveness of the iotSilo, we compared the iotSilo with other IoT enabling technologies. As shown in Table 1, the iotSilo is much better than other technologies in supporting heterogeneous devices. The iotSilo can support various heterogeneous devices, but WSN and M2M have the limit of device types. The dynamic function extension is easily supported by changing the configuration of agents in the iotSilo, but other technologies should modify components in the source level. In WSN and M2M, it is difficult for heterogeneous devices to communicate with each other because the technologies force the devices to have the same network connectivity. Comparing to it, the iotSilo successfully achieved the collaboration of devices on the different network connectivity by using the indirect communication containing conceptual directives. The iotSilo is superior to the conventional technology in resolving the heterogeneity issue of the IoT, but the iotSilo reveals some weaknesses of technology stability and the scalability due to its preliminary phase.

5.2. Federation of the iotSilos. From the evaluation, the scalability of the iotSilo should be improved. Since IoT is expected to be composed of at least several million devices, the scalability is essential property that all IoT platforms should support. The same numbers of agents corresponding to all devices have to be allocated in a single iotSilo, so the size of IoT depends on the capacity of the iotSilo. Generally, the methods of enhancing the scalability fall into two broad categories: scale-out and scale-up [28]. The scale-up is a simple way but it is not practical at all for the IoT environment because the number of devices can easily pass over the added resources. For this reason, the scale-out seems the only way to resolve the scalability issue of IoT. In the further study, we have a plan to design that multiple iotSilos can federate into a virtual single iotSilo in the manner of scale-out. Each iotSilo takes charge of each organization and delivers requests to other iotSilos when its devices want to collaborate with the devices belonging to the other iotSilo. Using this federation, the iotSilo can be extended to large scale easily.

5.3. Directory Service. In the current design, an agent has to know the target directive before calling it. That means
the directive should be contained in the behavior code. Since behaviors can be easily replaced in the configuration file, this structure does not severely affect the dependency between agents. However, if an agent can search other agents’ behaviors dynamically and understand semantically, the intelligent collaboration of agents may be possible. This kind of smart and automated collaboration of machines is another vision of the IoT. In the further research, we will study the semantic directory service with which every agent registers its behaviors and semantically searches others’ behaviors. For a long time, the related studies have been conducted in the field of the Semantic Web Services [29] and some meaningful methods such as SSWAP [30] have been produced. Therefore, it may not be difficult to make a directory service for IoT with the Semantic Web Services technology. Additionally, we also have a plan to create a new ontology which can describe agents’ behaviors and IoT services.

6. Conclusion

Although many visions have been presented and expectation about IoT is growing rapidly, there has not been any solution for IoT yet. There may be many issues to prevent the realization of IoT, but the heterogeneity is the most essential one. In order to resolve the issue, we suggested the agent service platform, iotSilo. It enables agents to communicate and collaborate to make IoT services on behalf of their corresponding devices. The structures of agent and protocol were designed to hide the heterogeneity of devices and to extend easily the devices’ functions. With Arduino, various IoT devices were made to emulate real world things and they were deployed in both Korea and Japan. In order to show the effectiveness of the iotSilo, several experiments were conducted. The results showed that the iotSilo enables devices to communicate and collaborate with each other regardless of their heterogeneous natures such as network types. The results also showed that the iotSilo enables agents to assemble or replace their device-dependent behaviors at runtime without corrupting the system.

The iotSilo has shown a new way to resolve the heterogeneity issue for IoT, but there is some room for improvement because it is still in the experimental stage. For the further studies, we have a plan to design a directory service that enables agents to register and find behaviors semantically. In addition, we will study the scalability of the iotSilo to support a huge number of devices.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of the paper.

References

[1] T. R. Regina, T. Tome, and C. E. Rothenberg, "Scenario of evolution for a future internet architecture,” in New Network Architectures, pp. 57–77, Springer, 2010.
[2] D. Bandyopadhyay and J. Sen, “Internet of things: applications and challenges in technology and standardization,” Wireless Personal Communications, vol. 58, no. 1, pp. 49–69, 2011.
[3] J. K. Ng, “Ubiquitous healthcare: healthcare systems and application enabled by mobile and wireless technologies,” Journal of Convergence, vol. 3, no. 2, pp. 15–20, 2012.
[4] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelflle, “Vision and challenges for realizing the internet of things,” Cluster of European Research Projects on the Internet of Things, European Commission, 2010.
[5] J. C. Augusto, V. Callaghan, D. Cook, A. Kameas, and I. Satoh, “Intelligent environments: a manifesto,” Human-Centric Computing and Information Sciences, vol. 3, article 12, 2013.
[6] L. Atzori, A. Iera, and G. Morabito, “The internet of things: a survey,” Computer Networks, vol. 54, no. 15, pp. 2787–2805, 2010.
[7] F. Mattern and C. Floerkemeier, “From the internet of computers to the internet of things,” in From Active Data Management to Event-Based Systems and More, vol. 6462 of Lecture Notes in Computer Science, pp. 242–259, 2010.
[8] L. Tan and N. Wang, “Future internet: the internet of things,” in Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE ‘10), pp. V5376–V5380, Chengdu, China, August 2010.
[9] E. Cho and S. Helal, “Expressive exceptions for safe pervasive spaces,” Journal of Information Processing Systems, vol. 8, no. 2, 2012.
[10] J. Yick, B. Mukherjee, and D. Ghosal, “Wireless sensor network survey,” Computer Networks, vol. 52, no. 12, pp. 2292–2330, 2008.
[11] K. Chang, A. Soong, M. Tseng, and Z. Xiang, “Global wireless machine-to-machine standardization,” IEEE Internet Computing, vol. 15, no. 2, pp. 64–69, 2011.
[12] M. Chen, J. Wan, and F. Li, “Machine-to-machine communications: architectures, standards and applications,” KSII Transactions on Internet and Information Systems, vol. 6, no. 2, pp. 488–497, 2012.
[13] M. Iqbal and H. B. Lim, “Poster abstract: a cyber-physical middleware framework for continuous monitoring of water distribution systems,” in Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys ‘09), pp. 401–402, November 2009.
[14] D. Wilcher, Learn Electronics with Arduino, Apress, 2012.
[15] The IPSO Alliance, “Technical Information,” 2010. http://www .ipso-alliance.org/technical-information.
[16] E. Fleisch, “What is the internet of things?—An economic perspective,” Auto-ID Labs White Paper WP-BIZAPP-053, 2013, http://autoidlabs.org/uploads/media/AUTOIDLABS WP-BIZAPP-53.pdf.
[17] O. Dubuisson, “Introduction to Object Identifiers (OID) and registration authorities,” France Telecom Orange, 2013, http://www.oid-info.com/doc/introductiontoobjectidentifiers(ODIs).pdf.
[18] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, “IOT gateway: bridging wireless sensor networks into internet of things,” in Proceedings of the IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC ’10), pp. 347–352, Hong Kong, China, December 2010.
[19] S. Hong, D. Kim, M. Ha et al., “SNAIL: an IP-based wireless sensor network approach to the Internet of things,” IEEE Wireless Communications, vol. 17, no. 6, pp. 34–42, 2010.
[20] Postscapes, “Internet of Things Platforms-Postscapes,” 2013, http://postscapes.com/internet-of-things-platforms.
[21] COSM, “Cosm—internet of things platform connecting devices and apps for real-time control and data storage,” 2013, http://www.cosm.com.

[22] Open.Sen.se, 2013, http://open.sen.se.

[23] H. S. Nwana and D. T. Ndumu, “A brief introduction to software agent technology,” in Agent Technology: Foundations, Applications, and Markets, pp. 29–48, Springer, Wooldridge, Mo, USA, 2010.

[24] K. Knoernschild, Java Design: Objects, UML, and Process, Pearson Education, 2001.

[25] M. Fowler, “Inversion of control containers and the dependency injection pattern,” 2013, http://www.martinfowler.com/articles/injection.html.

[26] C. Edward, “Message-oriented middleware,” in Middleware for Communications, Q. H. Mahmoud, Ed., pp. 1–28, Wiley, Chichester, UK, 2004.

[27] B. Snyder, D. Bosanac, and R. Davies, ActiveMQ in Action, Manning, 2011.

[28] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, “Scale-up x scale-out: a case study using nutch/Lucene,” in Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS ’07), March 2007.

[29] R. Studer, S. Grimm, and A. Abecker, Semantic web services: concepts, technologies, and applications, Springer, 2007.

[30] D. D. G. Gessler, G. S. Schiltz, G. D. May et al., “SSWAP: a simple semantic web architecture and protocol for semantic web services,” BMC Bioinformatics, vol. 10, article 1471, p. 309, 2009.