An Object-oriented Design and Push Web Server based Framework for Physical Object Interactions and Services

Runhe Huang, Kei Nakanishi, Jianhua Ma
Faculty of Computer and Information Sciences, Hosei University, Tokyo, Japan
Email: {rhuang, jianhua}@hosei.ac.jp
nakanishi@gs-cis.hosei.ac.jp

Bernady O. Apduhan
Faculty of Information Science, Kyushu Sangyo University, Fukuoka, Japan
Email: bob@is.kyusan-u.ac.jp

Abstract— **One of the substantial issues in ubiquitous computing is the automatic processing of information from real world objects and enabling their interactions in the background. This scenario requires a framework on which such information processing and object interaction can be supported. This article presents our research progress in developing a framework based on the object-oriented design approach and the use of a push web server. With the concept of object abstraction, an object can hide its internal structure from the outside world, which can make the object secure. Moreover, object interaction can be conducted via message exchanges, which makes the interface simple and standardized and the heterogeneous objects can be easily handled as well. Instead of using the traditional pull web server, a push web server, i.e., Comet, which runs on top of HTTP protocol is used to exchange messages. To this, object interactions can be operated smoothly and seamlessly in real time with shorter delay.**

Index Terms— **ubiquitous computing, pushing web service, object-oriented design, robot motion control, RFID, Comet**

I. Introduction

Different from cyber computing, ubiquitous computing emphasizes on adding computing ability to real world physical things such that they are able to process acquired information, communicate and interact with each other. In order to differentiate real world physical things with computing ability from those software entities known as e-things/e-objects, real world physical things with computing ability are called u-things/u-objects [1].

With the continuing miniaturization of electronic chips and electro-mechanical devices, there are more and more u-objects being developed ranging from handheld devices such as cell phones and PDAs, home appliances such as TVs and refrigerator, to ordinary goods. For example, a book can be attached with a RFID tag to store the book's related information such as the book's title, author's name, year of publication, ISBN number, etc. As a result, u-objects can be regarded as a special kind of physical entities that are able to store some data electronically and interact with each other via wired or wireless communications.

In contrast, software entity such as digital services, known as e-things/e-objects, which resides in either a local or a remote system, can be used to process u-objects related information by providing various specific service functions. A service performs some operations upon request and replies the request with the result. For example, an online bookstore may have a book information service which provides catalog searching, online purchasing, latest news, and so on. In fact, there are already many digital services, such as map service, navigation service, online banking service, and other services that are available via the Internet or other networks.

A ubiquitous system [2] generally involves many physical u-objects and digital services. They need to communicate and interact with each other in order to accomplish some assigned task(s). Therefore, both communication and interaction are important and necessary functions which a ubiquitous system has to support. To enable their efficient communications and interactions, it requires a middleware or framework. Due to the heterogeneity of u-objects and digital services, developing such framework is not an easy thing to do and there are many challenging problems to be solved.

There have been quite a number of developments [3][4][5][6] in this area based on the service-oriented architecture [7] in which everything is viewed as a service. This approach seems unnatural since some physical devices can hardly be regarded as services. In contrast, it is natural to regard everything as an object, and so it is appropriate to adopt the object-oriented design in which everything is viewed as an object [8]. This article proposes a framework based on the object-oriented design and Comet, also known as a push web server, which runs on top of HTTP protocol for message exchanges.

II. WHY OBJECT-ORIENTED DESIGN?

The concept underlying object-oriented programming was first introduced in the Simula programming language and Smalltalk was the first programming language to be called "object-oriented" [9]. Different from the traditional approach, the object-oriented approach regards everything in the real world and cyber worlds as objects and associates the data with operators on objects. It has the following advantages [10]: reusability, extendibility, scalability, and heterogeneity. The use of the object-oriented design is mainly based on the following two considerations: (1) both physical objects and digital services in the ubiquitous computing system can be viewed as objects; (2) it is a good idea to take the advantages of object-oriented design's reusability, extendibility, scalability, and heterogeneity. The object-oriented approach is therefore the appropriate one.

A. U-object Class vs. U-object Instance

In a ubiquitous computing system, real world physical objects like PDAs, RFIDs, sensors, and robots are considered objects, also called u-objects and the associated services are called u-services in order to emphasize they are in a ubiquitous computing context. Like an object in object-oriented programming, a set of u-objects can be defined as a software entity, called u-object class. The u-object instances can be generated from the u-object classes by assigning values to the variables, and the associated methods defined in the u-object class can perform their operations on the u-object instances. As an example, the partial code of RFID class, the generation of RFID instance *rfid1*, and the invocation of the method, startChat() are given below.

```
1. public class RFID {
2.   // variables
3.   private RFIDPhidget rfid;
4.   private Lingr lingr;
     // methods
     public RFID() {  super();  }
     public void initRFID()]{…...}
     public void startRFID(){……}
     private String getTag(String oe){……}
     public void startChat(){……}
  }
```

```
RFID rfid1 = new RFID();
rfid1.startChat();
```

B. Inheritance vs. Containment

Inheritance is a mechanism that provides a way to reuse existing classes, implementing small changes in behavior by overriding existing methods in the superclass or by adding new methods in the subclass [10]. For example, RFIDwithName, is a subclass that inherits all variables and methods of its superclass, RFID and includes additional method, toString() as given below.

```
public class RFIDwithName extends RFID {
   // additional methods
   public String toString(String oe){
      return getTag(oe);
   }
}
```

Object inheritance is referred to as the "IS" relation of a superclass with its subclass objects. For example, we can say that a RFIDwithName object IS a RFID object. Whereas object containment is referred to as the "HAS" relation of an object with other objects contained in the object. Such object is called a composite object. For example, a RFID object is a composite object since it has RFIDPhidget and Lingr objects in its class. The use of "HAS" relation permits the representation of complex structures, which is very important in a ubiquitous computing system since some objects like space object and room object are real complex objects that contain a number of heterogeneous other objects. It is one of the necessary features that any complex object should be composed and decomposed.

C. Encapsulation and Polymorphism

An object hides things from other objects and isolates its variables and methods from the external environment. This is called encapsulation property. This property protects an object's variables from corruption by other objects and hides the internal structure of the object so that interaction with the object is relatively simple and standardized and changes in the internal structure do not affect other objects [10]. For example, if we change the internal variable, RFIDPhidget, to other RFID reader, such as OtherRFIDReader, in the class RFID, it is rather easy to change Line 3 as,

```
        private RFIDPhidget rfid;
  →     private OtherRFIDReader rfid;
```

The change is completely unknown by external objects and does not affect the interfaces with external objects. An object is just like a black box to external objects and objects interact by means of message exchanges.

Different objects have the same method name but do things in different ways. This is called polymorphism characteristic. This characteristic hides different implementations behind a common interface [10]. For example, two different objects share the same method name, print(), but with different implementations in different object classes. One is for printing a text line and the other is for printing an image.

```
public interface PrinterInterface{public print();}

public class PrinterText implements PrinterInterface {
   // same method name but different implementation
   public void print(){……} // print a text line
}

public class PrinterImage implements PrinterInterface {
   // same method name but different implementation
   public void print(){……} // print an image
}
```

With this characteristic, we can use one higher-level object to invoke many different lower-level objects using the same message format to accomplish similar functions. Likewise new lower-level objects can be easily added with minimal changes to existing objects. These are the characteristics our platform requires to deal with heterogeneous objects and their interactions.

In conclusion, the inheritance mechanism makes objects reusable, the containment structure makes objects extendible and the system scalable. Furthermore, the encapsulation property makes objects secured and the interactions simple, and the polymorphism characteristic makes the platform heterogeneous.

III. Object interactions via messages On Comet

A. U-objects, V-objects, A-objects, and Interfaces

U-objects are real world physical objects in a ubiquitous computing system. V-object is an object instance that maps to a physical object. A-object is an abstract object instance that is generated from the A-object class that is an abstract class that contains the default variables, methods, and at least one abstract method. A V-object class is a subclass of the A-object class and inherits all variables and methods from the A-object class with additional necessary variables and methods and implements the abstract method(s) defined in the superclass. A V-object class can inherit the superclass, A-object class, and implement an interface as well, that is, implement the abstract method(s) defined in the interface [11]. For example, a robot for finding lost or misplaced objects. Below gives some partial source codes in Java to show the relations of V-object class, A-object class and interface class.

```
// A-object class
public abstract class AbstractRobot {
  private String name ;
  Position position;
  // constructor method for creating robot objects
  public void Robot(String name, Position positon){
     this.name = name; this.positon = position; }
  //default methods
  public String getName(){…}
  public Position getPosition(){…}
  //abstract methods
  public void robotAction();
}
```

```
// interface
public interface RobotInterface {
  //abstract methods
  public void sendTo(Message msg)
  public void receiveFrom(Message msg)
}
```

```
// V-object class
public class FindingRobot extends AbstractedRobot
                                implements RobotInterface{
   public void FindingRobot(String name, Position position){
      Super(name, position); }
   //implementation of the abstract methods
   public void robotAction(){……}
   //implementation of the abstract methods
   public void sendTo(Message msg){……}
   public void receiveFrom(Message msg){……}
}
```

One of the advantages of using the object abstraction is that it supports the separation of what operations are provided by the systems and components, and how their operations are implemented. The outside world does not care about the detailed implementation of the operators and just sends a message to the object. The object invokes the requested methods within it with the parameters in the received message. As a result, it hides the internal structure of objects and makes objects interaction relatively simple and standardized.

B. Message Exchanges on a Pushing Web Server

Objects interact by means of message exchanges. Once an object instance wants to interact with other object(s), it generates and sends out a message that includes the names of the sending objects, the receiving object, a method name of the receiving object, and other necessary parameters to execute the method. A message can only be used to invoke a method within an object but can not directly access the data of other objects. So there should be a place to exchange messages. Such place is the Comet web server, a push web server.

Comet is a new style web server with push technology [12]. It allows a web server to push data/events down to a browser without the browser explicitly requesting it. With the benefit that data/events are relayed in almost real-time and "wasteful" requests in a traditional web server are greatly reduced. As we all know, HTTP is a widely used communication protocol that is able to cross over machines and software boundaries as well as getting through a firewall. However, HTTP protocol does not maintain a persistent connection. A HTTP server's message is to be delivered to a client only when the client makes a request and after which the connection is terminated. In order to simultaneously detect a new event/message arriving at the server, a client has to repeatedly make polling requests to the server. If the polling is repeated too frequently, the computation and communication overheads increase greatly. If the interval between two polling requests is too long, all objects may receive messages with long latency, and the whole system could not respond requests or perform context changes in real time. Instead of repeatedly polling to get new events, a Comet application/client, when receiving a pushed message from a Comet server, can post the next request immediately so as to keep a persistent HTTP connection between the server and the client. The traditional HTTP application model versus the Comet application model is given in Fig. 1.

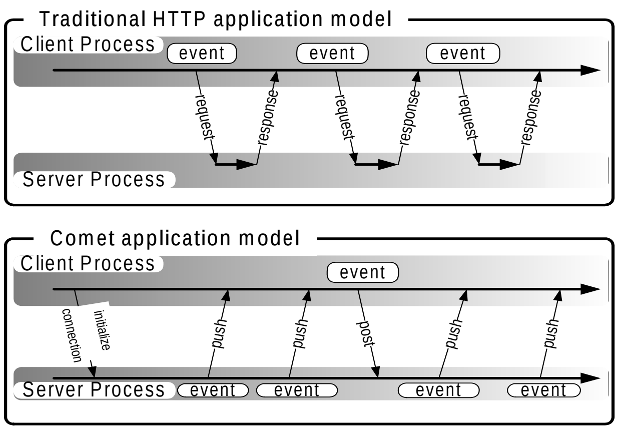


Figure 1. Traditional HTTP versus Comet

Some platforms have been supporting Comet. The Cometd [13] is a scalable Comet platform. The Jetty [14] has an implementation of the Cometd event server. In Tomcat, the new HTTP connector which uses NIO API has been added to Comet applications. The Resin [15] implements the Comet servlet API which enables the streaming communication. The Lingr [16] is the browser-based chat service using Comet for real-time message notification.

Figure 2 shows the interaction between two objects, i.e., a lost or misplaced object finding robot and a location service object, via message exchanges in the chat space on a Comet server which runs on top of HTTP protocol. The location service object posts a message requesting the location of the finding robot, and the Comet server pushes the event to its clients. Since the server and the clients are constantly connected, once the pushed event occurs, the client can almost simultaneously identify the event and take the corresponding action, such as return a message including the requested position of the finding robot to the chat space on the Comet server. The Comet server pushes the message to its clients, and the client as well as the location server can also almost simultaneously identify the returned message and receive it.

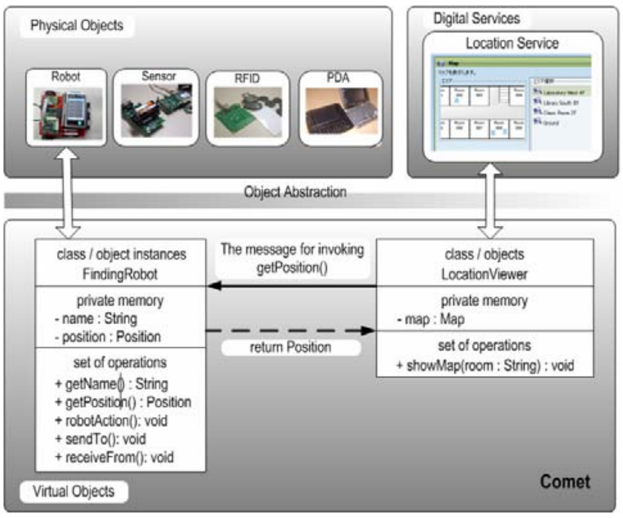


Figure 2. An example of two object interaction

In conclusion, with the use of Comet, i.e., a push web server on top of HTTP protocol, the proposed framework can take advantage of both the push technology and the HTTP protocol. That is, the server and its clients can keep a persistent HTTP connection and the communication protocol can cross over machine and software boundaries as well as go through a firewall.

IV. The 5-Layer Framework and its Architecture

The proposed framework for real world physical object interactions is mainly developed on the basis of the object-oriented design principles and the use of push web server, Comet. Object interactions are conducted with message exchanges in a chat space (Lingr) on Comet that runs on top of the HTTP communication protocol. The framework has 5 layers which start from the lowest layer, i.e., the physical network, then the HTTP protocol layer, 3-servers layer, and the object interaction layer which is below the highest layer, i.e., the application layer. Utilizing the advantage of HTTP protocol, the framework supports heterogeneous communications by crossing over any machines and software boundaries. With push technology in the web server, it enables persistent connections between the server and its clients so as to make object interactions with the least time latency to achieve almost real time object interactions.

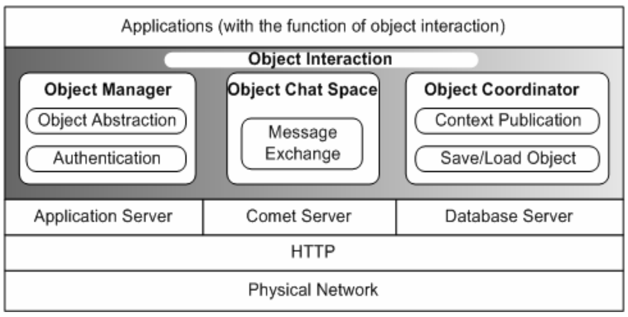


Figure 3. The 5-layer framework

The framework works on any machine and physical network so long as HTTP is supported. The object interaction layer consists mainly of three components: the object manager, the object chat space, and the object coordinator as shown in Fig. 3. They are designated for object management, object coordination, and message exchanges, respectively. The objects here refer to real world physical objects, such as PDAs, RFIDs, robots, sensors, and as well as those digital services. They are all implemented in Java.

Object Manager is a component for managing physical objects and digital services as well as their abstractions. Once an object logins, the authentication server verifies if the object is registered. If not, the login is refused or the object is reminded to make the registration first. If yes, the authenticated object is mapped to an object instance, i.e., v-object, of the corresponding object class, an abstraction of a set of objects. And then, the message exchanges with other object instances can be conducted in the common object chat space. The authentication server is implemented on Tomcat.

Object Chat Space is a common virtual place for message exchanges between objects. An object can send a message to and receive messages from others in the chat space. An object's message that usually requests other object(s) to do some operation(s) is sent to the object chat space, and the target object that receives the requesting message invokes an operation or some operations specified in the message. Other objects ignore this message. The object chat space, Lingr, is implemented on Comet, a push web server.

Object Coordinator provides two functions for object interactions. One is to analyze the received object's event message and publish semantic data to the object chat space where all other objects can notice the incoming data and take necessary response. Another function is to save or load an object with the object's state information. When a physical object or service logouts/leaves the framework, its corresponding v-object is saved in the object database. The object's configuration and state are preserved and the object is persistent for the continuous use next time. The context server for publishing context and analyzing event is run on Tomcat and the database server for storing objects is Derby.

The workflow of these components in the framework is shown in Fig. 4.

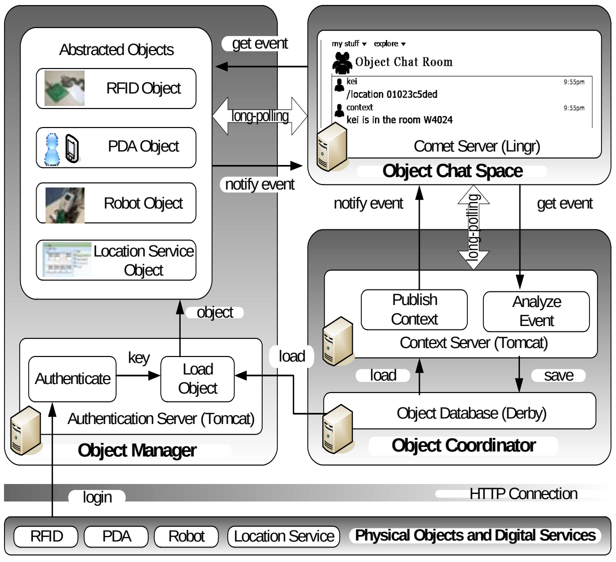


Figure 4. The system architecture and its workflow

V. CASE STUDIES

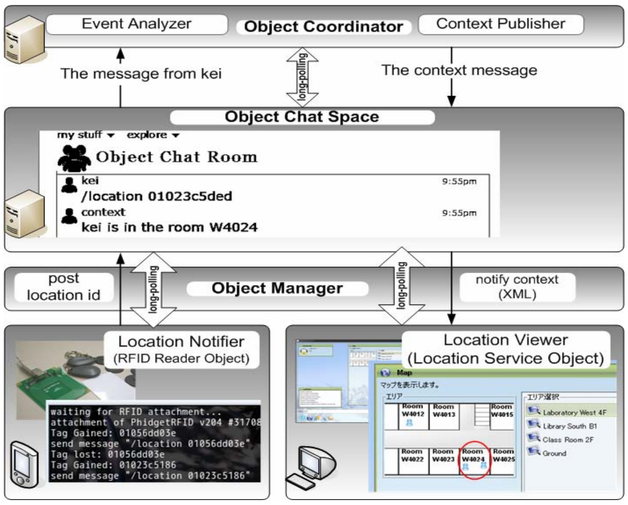
Applications which involve enabling various kinds of object interactions can be developed on the top layer of the framework. The following present three case studies in which we can see how two objects communicate via the object chat room in a user location tracking application, how an object can keep its persistent state and for another object to continue the unfinished task, and how a number of objects interact in a robot motion control application.

A. *Case 1: User location tracking*

This case study is to show how the system tracks a human object and how the location view object gets the human object's location and display it on the map.

The situation: The human, named Kei Nakanishi, is in room w4024. He is attached with a RFID tag in which his name, "kei" and tag'ID are written. A user with PDA can dynamically see the icon representing him located in the map on the display screen as shown in Fig. 5.

The workflow: As shown in Fig. 5, the RFID tag object is detected by the location notifier, an RFID reader object. The information contained in the RFID tag is read and sent to the object chat room.



```
kei                                   9:55pm
/location 01023c5ded
context                               9:55pm
kei is in the room W4024
```

```
waiting for RFID attachment...
attachment of PhidgetRFID v204 #31708
Tag Gained: 01056dd03e
send message "/location 01056dd03e"
Tag lost: 01056dd03e
Tag Gained: 01023c5186
send message "/location 01023c5186"
```

Figure 5. The user location tracking

The message regarding the "kei" object is, in this example, "/location012023c5ded". The message is pushed and the object coordinator obtains the message which is then analyzed by the event analyzer and translates it into a context message, "kei is in room w4024". The context publisher posts the context message, and Comet pushes the messages into the object chat room. The location viewer object identifies the context message, pulls the context message in XML format, parses the XML file and extracts the human's object name, "kei", and the location, "w4024". The icon of the human object is shown in the map on the display screen. If the user clicks on the icon, the detailed information about the human object is displayed in the right and bottom corner of the location viewer window.

B. *Case 2: Continuation of a u-object's activity*

This case study is to explain how a u-object maintain and continue its state and how its designated task can be continually implemented by itself or by other object.

The situation: There are two robots, named robot-A and robot-B. Robot-A is assigned a task, i.e., going from its current position to a specified position such as the coordinates of the position, (100, 50) to perform a task. However, when robot-A is obstructed by a wall along the way, it is not possible or not worthy for robot-A to go around to reach the destination or robot-A may run out of battery and thus can not continue its task, and has to terminate for the time being. In such cases, it may be better to let other robot such as robot-B which is closer to the destination to continue robot-A's task or let robot-A to continue its task after recharging its battery. No matter which case, it is necessary to store robot-A's state when it is terminated and post its unfinished task to the object chat room so that other robots or robot-A itself can continue the task.

The workflow: As shown in Fig. 6, two physical robot objects are mapped to two virtual robot objects. Partial information like the robot current state and two methods for each virtual robot object are presented in the figure.

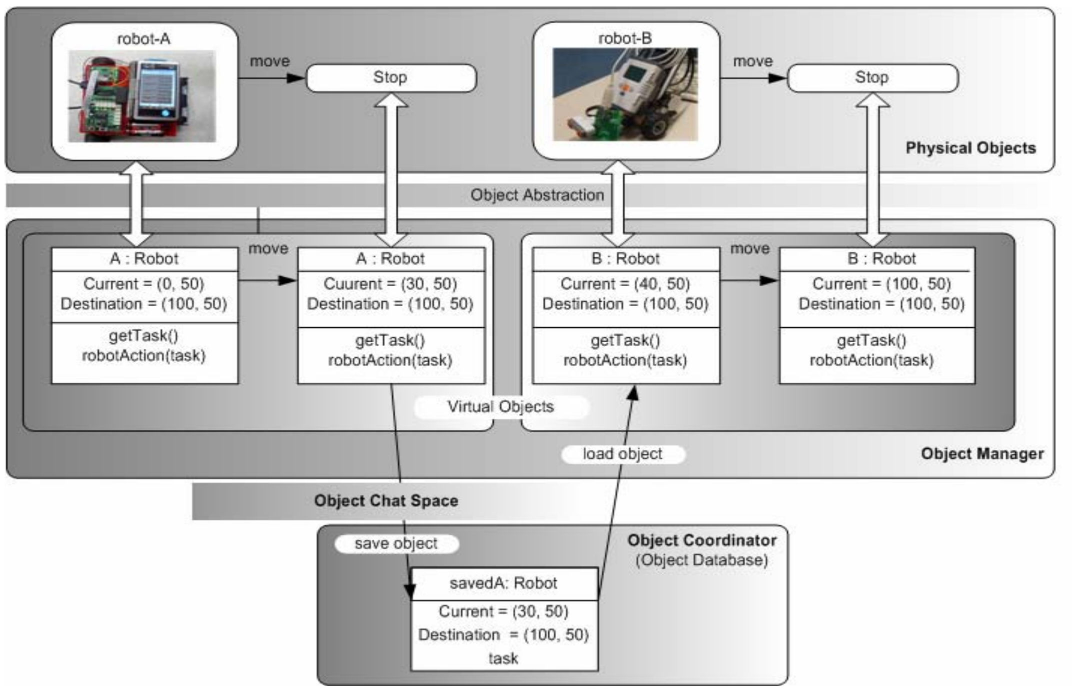


Figure 6. The continuation of robot-A's activity by robot-B

When robot-A encounters a wall and has to stop, its current state, the destination, and the task to be performed are saved as a save object which is then sent to the object coordinator via the object chat space, and stored in the object database. The saved object with respect to robot-A can be loaded by robot-A later or loaded by other object such as robot-B which is requested to continue robot-A's task. It is observed that the methods used to continue a u-object's activity can be similarly applied to keep objects persistence at a certain period of time in any application.

C. *Case 3: A Robot's Motion Control*

This case study is to demonstrate how an object and a number of service objects conduct communications and interactions in order to achieve the task of controlling the robot's motion.

The situation: The robot, ROBODESIGNER RDS-X03 Platform+CDE, is equipped with a Phidget RFID reader and a Zaurus SL-C3100 PDA. A number of RFID tags are distributed in the surroundings. When the robot moves, the RFID reader can detect a nearby RFID tag and read the data from it. With the received data, the robot's current position can be calculated. The PDA acts as a communication gateway and a data/program processor. It communicates with all other services which can be residing in a site or distributed in different sites, processes the receiving data/program, and then send the resulting data to the robot for controlling the robot's motion. The service objects include LookupService, ShellService, MapService, RouteService, and CompileService. These services function as their names implies. LookupService provides a list of available services. ShellService provides a way for users to type command line. MapService replies a map upon request. RouteService searches for a route from start to destination. CompileService generates the motion data by converting the route data to C source code and then compiling it. The resulting S19 file is for the robot motion control.

The workflow: The robot logs in to the system via the PDA and the services register to the system. No matter whether it is a robot object or a service object, each object is mapped to a v-object. All v-objects reside in the object management module and interact with each other in the object chat space. The sequence of their interactions in the object chat space is given in Fig. 7.

After the robot object and the service objects log in or register to the system, a user can see the robot icon and get a list of services from the interface window. As shown in Fig. 7, the sequence of object interactions in robot motion control is finding the robot's position, calculating the route, and sending the motion code to the robot. These are described in the following steps:

① The interface object sends a message, calling the method getServiceList() to the LookupService object and receive the reply message of serviceList.

② The user can type in a command line like "TestMessage | RouteService | CompileService | Robot" and then the interface sends the command line and the message to call the method, connectServices() to the ShellService object.

③ The ShellService object sends a message to call the method, getLocation() to the robot object.

④ After receiving the robot's location data from the robot object via the PDA, the ShellService object sends a message to the RouteService objects which includes the robot's location data, the destination data, and the message to call the method, searchRoute().

⑤ The RouteService object then sends a message to call the method, getMap() with the location data to the

MapService object, and then receives the returned map in which the location of the robot object icon is indicated.

⑥ The RouteService object invoke its own method generateRoute(), and returns the generated route back to the ShellService object.

⑦ The ShellService object sends a message to call the method, outputSource() with the received route data to the CompileService object.

⑧ The CompileService object invokes its own method, compileSource() with the generated source code, and then returns the motion data to the ShellService object.

⑨ Finally, the ShellService object sends a message to call the method, move() with the received motion data to the robot object, and then the robot starts to move according to the received motion data.

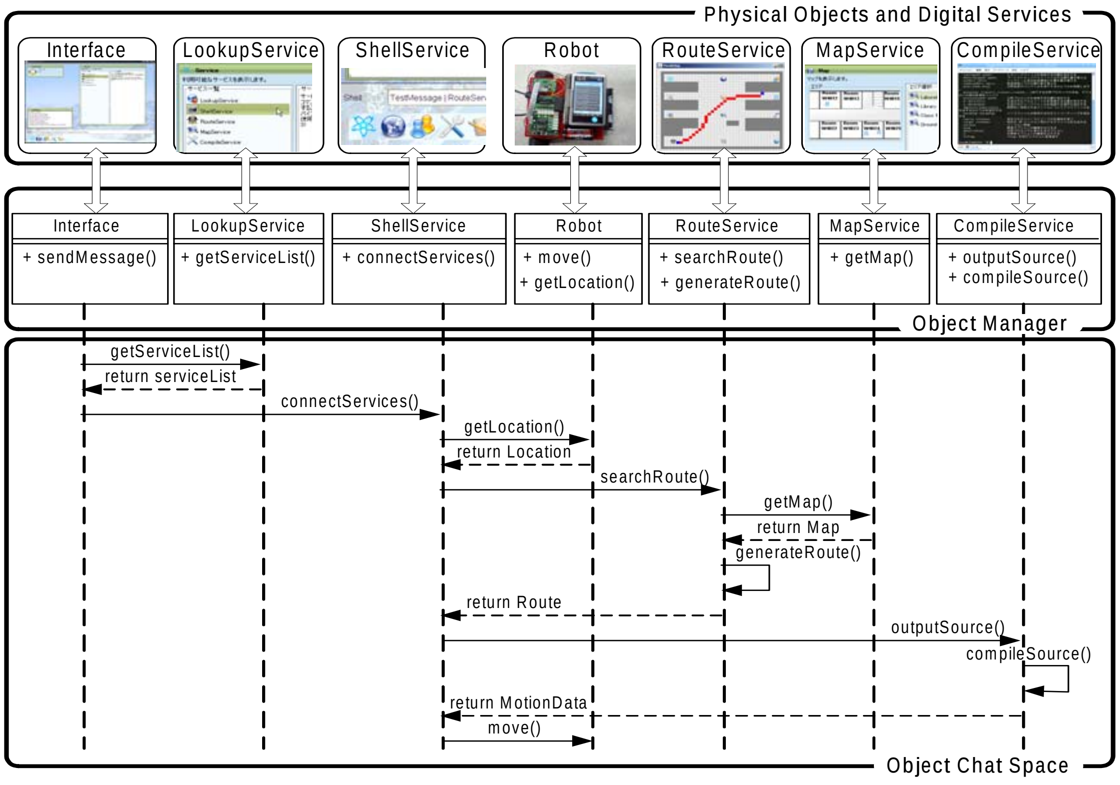


Figure 7. The interaction flow in the robot's motion control

From the above three case studies, it can be seen that the framework can host a variety of heterogeneous physical objects and digital services and support their interactions. With the object-oriented design, the implementation of the interface of objects interaction becomes simple and standardized, that is, via message exchanges on Comet. With respect to the internal structure of an object, it is hidden from the outside world so that it is protected from corruption and any changes in the internal part do not affect other objects and the framework. This proves that the object-oriented design is an appropriate approach to realize the framework.

VI. Concluding Remarks

This article described a framework based on the object-oriented design and a push web server for object communication and interaction in the real world and cyber world. The object-oriented design was proven to be the appropriate approach which enables the framework to handle heterogeneous objects and simplify their interactions just by exchanging messages. The interface for the objects communications and interactions became simple and standardized. Instead of using the traditional pull web server, the Comet, also known as a pull web server, was used to host the object chat room which is a place for objects to exchange messages. Using push web server that runs on top of HTTP protocol makes message exchanges of objects more efficient and shorter delay.

Although this framework is platform and network independent, and provides heterogeneous objects interactions using HTTP messages, a general event analysis mechanism for objects to easily understand semantic meanings of incoming messages should be further studied and developed. In the current framework, all objects exchange messages via one Comet web server. Although one server is sufficient for the above mentioned three case studies, it may encounter some scalability problems especially in a large scale ubiquitous system in which a huge number of objects are involved. One possible solution is to use a group of distributed Comet web servers in a scalable, hierarchical, and collaborative structure. Derby, also known as a relational database, was used in the object coordinator for storing and managing v-objects. In fact, instead of using a relational database, it is believed that an object-oriented database such as db4o is more appropriate database to use.

REFERENCES

[1] J. Ma, “Smart u-Things – Challenging Real World Complexity”, IPSJ Symposium Series, Vol. 2005, No. 19 (2005) 146-150.

[2] J. Ma, L. T. Yang, B. O. Apduhan, R. Huang, L. Barolli and M. Takizawa, “Towards a Smart World and Ubiquitous Intelligence: A Walkthrough from Smart Things to Smart Hyperspaces and UbicKids”, *International Journal of Pervasive Comp. and Comm.*, 1(1), March 2005.

[3] Ikuo Yamasaki, Kouji Yata, Hiroyuki Kaeomichi, Akihiro Tsutsui and Ryutaro Kawamura, “Security Functions of a Distributed Network Middleware CSC on OSGi Frameworks”, The Institute of Electronics, Information and Communication Engineers, *IEICE Technical Report. Information Networks*, Vol.105, No.113, pp. 35-40, 2005.

[4] T. Kawamura, K. Ueno, S. Nagano, T. Hasegawa and A. Ohsuga, “Ubiquitous Service Finder - Discovery of Services Semantically Derived from Metadata in Ubiquitous Computing”, Proc. of 4th International Semantic Web Conference (ISWC 2005), 2005.

[5] J. Ma, R. Huang, and R. Nakatani, “Towards a Natural Internet-Based Collaborative Environment with Support of Object Physical and Social Characteristics”, *International Journal of Software Engineering and Knowledge Engineering*, pp37-53, No. 2, Vol. 11, 2001.

[6] T. Kawashima and J. Ma, “TOMSCOP -A Synchronous P2P Collaboration Platform over JXTA”, IEEE CS Proceeding of the International Workshop on Multimedia Network Systems and Applications (MNSA’2004), Tokyo, Japan, March, 2004.

[7] M. P. Papazoglow and D. Georgakopoulos, “Service Oriented Computing”, in *Communications of the ACM,* Vol.46, No.10, pp.25-28, October 2003.

[8] J. Hunt, *Smalltalk and Object Orientation - An Introduction*, Springer, 1997.

[9] Wikipedia, “Object-oriented Programming”, http;// en.wikipedia.org/wiki/Object-oriented_programming

[10] W. Stallinngs, *Operating Systems – Internals and Design Principles,* the Third Edition, Prentice-Hall International.

[11] T. Sintes, “Abstract classes vs. interfaces”, *JavaWorld.com*, 04/20/01.

[12] A. Russell, “Comet: Low Latency Data for the Browser”, Continuing Intermittent Incoherency, http://alex.dojotoolkit.org/?p=545, November 2007.

[13] Cometd.org, http://cometdproject.dojotoolkit.org/

[14] Jetty Web Server, http://www.mortbay.org/jetty/

[15] Server-push servlet, http://www.caucho.com/resin/examples/servlet-comet/

[16] Lingr, http://www.lingr.com/.

Runhe Huang received her B.Sc. in Electronics Technology from National University of Defense Technology, China, in 1982, and her PhD in Computer Science and Mathematics from the University of the West of England, UK, in 1993. After receiving her PhD, she worked in the University of Aizu, Japan, for 7 years and has been working in Hosei University, Japan, since 2000.

She is a Professor in Faculty of Computer and Information Sciences at Hosei University. Her research fields include Computer Supported Collaboration Working, Artificial Intelligent Applications, Multi-Agent Systems, Multimedia and Distributed Processing, Genetic Algorithms, Mobile Computing, Ubiquitous Computing, Grid Computing, and Ubiquitous Intelligence.

Professor Huang is the member of IEEE and ACM. She has served as an editor board member of the Journal of Ubiquitous Computing and Intelligence (JUCI) and the Journal of Autonomic and Trusted Computing (JoATC), as a guest editor of a number of the special issues and as a PC co-chair/PC member for various international conferences.

Kei Nakanishi received his B.S. and M.S. degrees in computer and information sciences from Hosei University in 2005 and 2007, respectively. He is currently a software engineer in Research Institute of System Planning, Japan. His research interests are web service, networked system administration, ubiquitous computing, and mapping technology between real and cyber worlds with using various sensors, RFID tags and other devices.

Jianhua Ma received his B.S. and M.S. degrees from National University of Defense Technology (NUDT), China in 1982 and 1985, respectively, and the Ph.D degree from Xidian University in 1990. He has joined Hosei University since 2000.

He is a professor in the Faculty of Computer and Information Sciences at Hosei University. His research from 1983 to 2003 covered coding techniques,, data/video security, speech recognition and synthesis, multimedia QoS, graphics rendering ASIC, CSCW, multi-agents, mobile web service, P2P network, etc. Since 2003 he has been devoted to what he called Smart Worlds (SW) pervaded with smart/intelligent u-things, and characterized by Ubiquitous Intelligence (UI) or Pervasive Intelligence (PI).

Professor Ma Professor Huang is the member of IEEE and ACM. He has published over 150 referred papers in journals and conference proceedings. He is the Co-EIC of JMM, JUCI and JoATC, and Ass. EIC of JPCC.

Bernady O. Apduhan received his B.S. Electronics Eng’g. degree from MSU-Iligan Institute of Technology (MSU-IIT), Philippines. He studied M.S. Electrical Eng’g. at the University of the Philippines, Diliman Campus, and received his M.S., Ph.D. in Computer Science degrees from Kyushu Institute of Technology (KIT), Japan.

He was a faculty member at the Dept. of Electronics Engineering, MSU-IIT, was a research associate at the Dept. of Artificial Intelligence, KIT, and is currently an Associate Professor in the Dept. of Intelligent Informatics, Kyushu Sangyo University, Japan. His current research interests include cluster/grid computing, ubiquitous computing and intelligence, and mobile/wireless computing and applications.

Professor Apduhan served as an editorial member of the Transactions of IPSJ-DPS, Int’l. Journal of Business Data Communication and Networking, Journal of Ubiquitous Computing and Intelligence Computing, and as a reviewer in other respected journals. He also served as executive steering committee member, PC Chair/PC Co-chair/PC member, and reviewer in various local and international conferences. He is a member of IPSJ-SIGDPS, IPSJ-SIGHPC, and a professional member of ACM and IEEE-CS.