Journal of Information Hiding and Multimedia Signal Processing ©2014 ISSN 2073-4212
Ubiquitous International Volume **5**, Number **1**, January **2014**

A Framework for Prototyping Telecare Applications

Feng-Cheng Chang
Department of Innovative Information and Technology
Tamkang University
180 Lin-Wei Road, Chiao-Hsi Shang, I-Lan County, 262, Taiwan
135170@mail.tku.edu.tw

Hsiang-Cheh Huang*
Department of Electrical Engineering
National University of Kaohsiung
700 University Road, Kaohsiung, 811, Taiwan
*:Corresponding author
hchuang@nuk.edu.tw

Received June, 2013; revised September, 2013

Abstract. *Telecare is the term for providing remote care to less able people such as elderly people and babies. A elecare application may be composed of several software and hardware components. The typical deployment structure is a distributed environment for hosting these interconnected components. In this paper, we design a software framework that is suitable for prototyping a telecare application. The framework is a realization of the service oriented architecture (SOA). Therefore an application is implemented as a few services. The Extensible Messaging and Presence Protocol (XMPP) network is the infrastructure for service collaboration. The flexibility of the XMPP allows dynamic configuration of the services and thus good for the prototyping purpose. At the end of the paper, we demonstrate a telecare application based on our framework. The analysis, design, and implementation are described to show the effectiveness and feasibility of the framework.*

Keywords: Telecare, Extensible Messaging and Presence Protocol, XMPP, Service Oriented Architecture, SOA

1. Introduction.

The trend of prolonging human life expectancy indicates that there are more and more elderly people. Elderly people may need different assistant technologies to make their life better. In addition to the utilities that helps them physically, monitoring technology around them would reduce the safety concerns [1, 2, 3]. A typical caring scenario is that the caretaker stays nearby the elderly people and keeps an eye on them. This inevitably interferes to their daily life. A solution to this situation is to adopt the telecare technology. Telecare is the concept of offering remote care to help old and physically less able people. According to the context of the use case, different assumptions and requirements lead to various application scenarios. There have been different researches related to this concept. Some of them focus on the network, some on the data analysis, some on the software model, etc.

In a telecare system, sensors may be integrated to monitor the environment or the biomedical conditions [4]. Depending on the available resources, a telecare system may be configured as various interconnected software/hardware components [5, 6, 7, 8]. For example, it would be a dedicated hardware that detects fall events and automatically sends short messages to the caretaker's mobile phone; or computer software that continuously analyzes the frames grabbed from the video recorder and sends fall alerts to the caretaker's messenger program. To determine the optimal configuration of the application, it is convenient if we have an environment for evaluating different combinations of the components. The more prototypes we evaluate, the more confidence we have on the released version. Therefore, the major concern of the prototyping environment is the flexibility to integrate various kinds of components. The performance of the intermediate prototype systems is typically not the first priority.

According to the properties of telecare applications, distributed communications among components are frequently encountered. Therefore, a telecare application can be thought of as the integration of necessary distributed service components. This is a good match to the concept of service-oriented architecture (SOA) [9, 10, 11, 12, 13]. In this paper, we will develop a telecare application as an SOA example based on the extensible messaging and presence protocol (XMPP) [14, 15, 16].

The paper is organized as follows. In Sec. 2, we discuss the structure of the prototyping framework and briefly describe the major concepts. In Sec. 3, we present the detailed design of the framework including the fundamental service. Then we choose a telecare scenario and demonstrate the development process of the application with this framework in Sec. 4. A few design concerns of the framework are discussed in Sec. 5. At the end, we conclude our work in Sec. 6.

2. Structure of the Prototyping Framework.

In this section, we identify the requirements of the prototyping framework for telecare applications. The technologies related to the structure of the framework are also briefly introduced.

2.1. Overview of the requirements.

Telecare technology is important for physically less able people. Generally speaking, we may deploy different kinds of sensors that detect different events in the monitored environment. When a potentially dangerous event occurs, the system notifies the caretaker to handle the event. A telecare system may also act as an informative reminder to the participants. By the help of telecare system, different caretakers would be involved: a professional person, a family member, or another helping system. No matter they reside locally or remotely, they are associated with the monitored people by the system.

A production-level telecare system should be compact and reliable, especially at the monitored side. Therefore, commercial telecare systems are usually implemented with customized (and sometimes proprietary) network and sensor hardware. Compactness may imply inflexibility. On the one hand, the customized and hardwired components make it difficult to change the configuration of the application. On the other hand, a high-precision or high-quality telecare system is sometimes not affordable. Therefore, we need an alternative architecture for prototyping the applications and/or reducing the costs. Fortunately, PC, wired and wireless LAN, and hand-held devices are commonly available. What we need is a software framework for developing the applications. The framework is a platform for integrating distributed services (including sensors), and it is better to be independent of programming languages. In our previous study [15], we found that a protocol centric framework is neutral to the underlying implementation. When the proper protocol is chosen, a service-oriented architecture can be deployed without concerning much about the programming languages. In the following sections, we will introduce the fundamental concept and the protocol used in our design.

2.2. **Service oriented architecture.** As described by I. Sommerville *"Serviceoriented architectures (SOAs) are a way of developing distributed systems where the system components are standalone services executing on geographically distributed computers [17]."* In terms of software design, it is the concept that an application is composed by several smaller pieces called services. The evelopment approach emerges as a result of network technology improvement and wide availability. Generally speaking, a service is a standalone software component that exchanges information with the other services to accomplish a certain task. An application in this sense includes the algorithm to locate all the necessary services, to associate services together, and to start the data or event flow to accomplish the task.

SOA can be implemented in different forms, such as the mature Common Object Request Broker Architecture (CORBA) [18] or web-based services. To integrate services, a broker for centralized management or agent-based technology for distributed management is necessary. The service management entity provides the functionality for registering and discovering a service. To manage services, some standard description specification should be used. For example, Web Services Description Language (WSDL) [19] is an XML based document for describing the features of a service. The matching algorithm of a service is not standardized, and it depends on the design of the application.

2.3. **Extensible messaging and presence protocol.** Extensible Messaging and Presence Protocol (XMPP) [16] is an open specification for delivering messages and multicasting presence status. It was developed as a messenger protocol called *jabber*. In an XMPP network, a client connects to a server and uses an e-mail like string (called bare JID) for authentication. The specification allows multiple logins of the same client, and thus the client may suggest an additional resource string for distinguishing different logins. Once the login process succeeds, the server responds the complete identifier (full JID) to the client. The suggested resource string may be overridden by the server as long as the result full JID is unique. The formal identifier of a client is the string `<client id>@<server>/<resource>`. When sending a message, the flow of the message goes through the source client, source client's server, the target client's server, and the target client. The connections among servers form the server federation. An XMPP server is required to deliver the message as soon as possible, and this is sometimes referred to the real-time property of the XMPP.

When an incoming message is addressed by a bare JID, the server chooses the default login instance as the target. To determine the default message receiver, a priority is associated with each login. The integer values from 0 to 127 are the potential receiver, and the login with the highest one wins. The integer values from -1 to -128 can only be addressed by the full JID. It is possible to change the priority at run-time, which implies that we can dynamically switch the default message receiver if necessary.

The access control issues are addressed in the core XMPP specifications. When a client initiates a connection to the server, an encrypted channel is established as requested. The client then authenticate with the server for login. Once the authentication is approved, the client sends the *initial presence* to notify the server for processing its presence. The initial *roster* (the subscribed friends list) is then fetched. Based on the roster, the presence of a particular friend is updated by the server. If message exchange should be blocked, this can be done by setting the *privacy list*. The server will deliver or reject the message according to the recipient's privacy list.

3. **Framework Design.** In the previous section, we briefly discussed the structure of the XMPP-based SOA framework. The run-time infrastructure is an XMPP network.

To make it an SOA platform, we need to enforce some common behaviors for managing services. In this section, we describe the core design of the framework. In Sec. 3.1, we design the fundamental service that provides the common functionality for further extension. In Sec. 3.2, we discuss the typical ways to use the fundamental service.

3.1. **Design of the fundamental service.** As the open-source community and many software vendors adopted the XMPP core specification, more and more extensions have been added to support various functionalities, such as binary object transfer, service discovery, audio/video streaming, etc. In our previous work, we found that XMPP is suitable to be the nderlying communication channel for SOA. According to the implementation experiences, we found that telecare devices and services vary from application to application. To design a common framework for different kinds of telecare applications, a fundamental service that defines the essential communication messages among all the derived services is necessary.

A service can be implemented in three forms: as a normal client (called a *bot*), as a component, and as a server extension (plugin). If performance is critical, the choice is server extension. However, it is not flexible because a server extension depends on the server implementation. Our goal is to develop a framework that allows integration of standalone services. Therefore, the former two approaches are better for this purpose. A client and a component are similar in that they should be authenticated before being available to the others. Most of the data exchange mechanisms, such as messaging and events, are handled in the same way. One of the differences is that a component has its own sub-domain and do not have a roster. With a sub-domain on a server, a component becomes more accessible by the other services. A component is also trusted by the server, and has permissions to control restricted resources. Therefore, it is suggested that a service is developed as client-based and later migrated to component-based when necessary. In the following discussions, we focus on the client-based design because it meets the requirements of the framework.

In most of the SOA implementations, service description and service discovery are necessary for dynamic application composition. They can be omitted if the SOA application is finalized or statically bound to dedicated services. Prototyping requires flexibility in dynamic configuration of components. Therefore, the fundamental service should enable the discovery mechanism by default. The XMPP Extension Protocol 0030 (XEP-0030) provides the functionality of discovery protocol. A client may request a list of available entities on the server, and queries the features announced by each entity. A $feature$ is a free-format string, and we can use a feature to describe one or more capabilities of the service. An identifier to locate the run-time service instance is required. Although the full JID is the natural service identifier in an XMPP network, the resource string is determined by the server (the server may override the resource string during the login process). In other words, a client may not have the same full JID for every session. A reliable way is to use a feature string as the application-specified identifier of the service instance, and then determine its full JID in the discovery process. The procedure to instantiate a fundamental service is specified as follows:

1. Connect to the server with the login JID and password.
2. Turn autosubscription on to make its presence automatically available to the others This step is optional if the service is intended for private use Note that turning off it does not affect the current subscribers on the roster Service owner should manually grant the subscription request when the feature is off.
3. Enable the XEP-003 discovery mechanism.
4. Wait for the session start event.

5. Announce the features including the bot identifier.
6. Send the initial presence with priority -1 so that it never receives messages addressed by the bare JID.
7. Fetch the initial roster Afterwards the service starts receiving the presence events from the other entities The roster is also the list of entities that are available for service discovery.

3.2. **Using the fundamental service.** There are a few issues of how to use the fundamental service. Some of them depend on the application scenario. Some are affected by non-technical factors. In this section, we focus on the technical issues. Non-technical ones are discussed in Sec. 5.

A technical issue is how to adopt the fundamental service in a specific service implementation. Generally speaking, either extension or composition is feasible. The former approach inherits the fundamental service, adds service specific functions, and accesses the underlying messaging mechanisms directly. The latter approach treats the fundamental service as a powerful communication module. The service specific functions are designed as a separate module. When starting the service, the specific functions are bound to different event handlers. The trade-off of the two approaches is that the ease of implementation and the flexibility of asynchronous processing. By the extension approach, it is straightforward to implement a function as a sequential algorithm, but the drawback is the lack of asynchronous event processing. On the contrary, it is natural to support event-based and concurrent processes by the composition approach. Of course,the overhead of asynchronous processing is inevitable.

For pure software services, we may decide to use extension or composition according to the concurrency requirements of the functions. For hardware related services, such as sound sensors, composition is better than extension. The reason is that a hardware circuit is typically standalone data source or sink. The asynchronous model is good for efficient processing. Otherwise, the CPU would waste a lot of cycles waiting for the input/output. The concept of developing a hardware related service is that we wrap the hardware interface inside the service. The wrapper service is responsible for processing the information between the hardware and the software interface.

4. **Application Example.** Based on the service framework described in Sec. 3, we may develop various kinds of services that communicate over the XMPP network. In this section, we describe how a telecare application is developed on the framework from scenario to implementation.

4.1. **Scenario.** Suppose we would like to prototype a telecare application without production-grade hardware. To achieve the goal, we need to write a program that communicates with commonly available computer peripherals and electronic devices. The telecare environment helps the caretaker (e.g., a family member at work) to monitor and remind the elderly people at home. The following functions are provided by the application:

- Fall detection in the bathroom and the living room.
- Reminder for taking medicine.
- Notify the caretaker when a fall is detected or the medicine is not taken.

4.2. **Services.** According to the aforementioned scenario, the functionality can be achieved by four functional parts: fall detection, reminder, notification sender (for both fall detection and reminder), and notification receiver (the caretaker). In this section, we identify the necessary services and the deployment structure.

For fall detection, a component that monitors human body movement and posture is required. The detection algorithm depends on the sensors involved in the mechanism. For instance, some algorithms are based on accelerometer, some are based on video analysis [20], and some are based on image analysis. In general, active detection strategies require the target person to be equipped with sensing devices, and passive strategies deploy sensing devices in the environment. It is obvious that the former is more accurate than the latter. However, passive strategies are less inconvenient to the monitored person. In this example, we choose image analysis approach for demonstration purpose. It requires less computation and data exchange bandwidth than video-based approach. The issue is when to take a shot. Periodically grabbing frames for analysis is straightforward but inefficient. A solution is to use a sensor that detects abrupt movements and triggers the image analysis. Based on the heuristic design rule, a service typically provides dedicated simple function(s). Therefore, we implement the image analysis function by the image capture service and the fall image analysis service. The capture service can be implemented by a low-profile computer with a camera module. The fall analysis service is pure-software and computation intensive. It is better to deploy the service to a powerful computer. Considering the variation of communication and computation latency, the image is pulled from the capture service rather than pushed to the analysis service. When the monitored person falls, an abrupt movement is detected, the analysis service is notified, the image is pulled from the capture service, and the notification service is invoked.

The notification sender and receiver, though separate components, should be considered as a whole. The combined result is a mechanism that matches the popular publishersubscriber design pattern. A service may publish a few events to the others. The receiver of an event is subscription-based. Hence, the notification sender is a service that provides subscription management and event dispatching. Using the fall detection scenario as an example, the fall image analysis service registers its notification type to the notification service. All the caretakers' programs send subscription requests to the notification service at startup. The notification service keeps a list of the fall event subscribers internally. When a fall is detected, the fall analysis service publishes the situation by sending a message to the notification service, and the service forwards the message with some additional attributes (will be described in Sec. 4.3) to all the subscribers (caretakers).

The medicine-taking reminder can be achieved by combining three services: the scheduler service that generates a medicine-taking event at a specified time, the reminder service that generates audio-visual signals to notify the target person, and the medicine-taken service that stops the reminder (right after medicine is taken) or notifies the caretaker (when medicine is not taken after the specified timeout). To simplify the mplementation, the medicine-taken detection is done by monitoring the status change of the medicine box's lid. Therefore we replace the service name with button service. The reminder works as follows. The scheduler service sends an event to trigger the reminder service to generate the audio-visual signals. The reminder service also starts a countdown timer and waits for a stop event. If the button service detects a lid-open, it sends a stop event to the reminder service to terminate both the signal and the timer. If the stop event is not received after the specified timeout, the reminder service stops the signal and publishes an event through the notification service.

The deployment structure of the services is shown in Fig. 1. In this configuration, we do not host our own XMPP server. Instead, we route all the messages through the Google-talk server. This is feasible because most of our services requires infrequent message passing. The image analysis service is the only one that pulls bulk data from the capture service. Because the two services are located in the home network, we use the

XMPP messaging as the control channel and transfer the image data via a dedicated TCP connection between the two services.

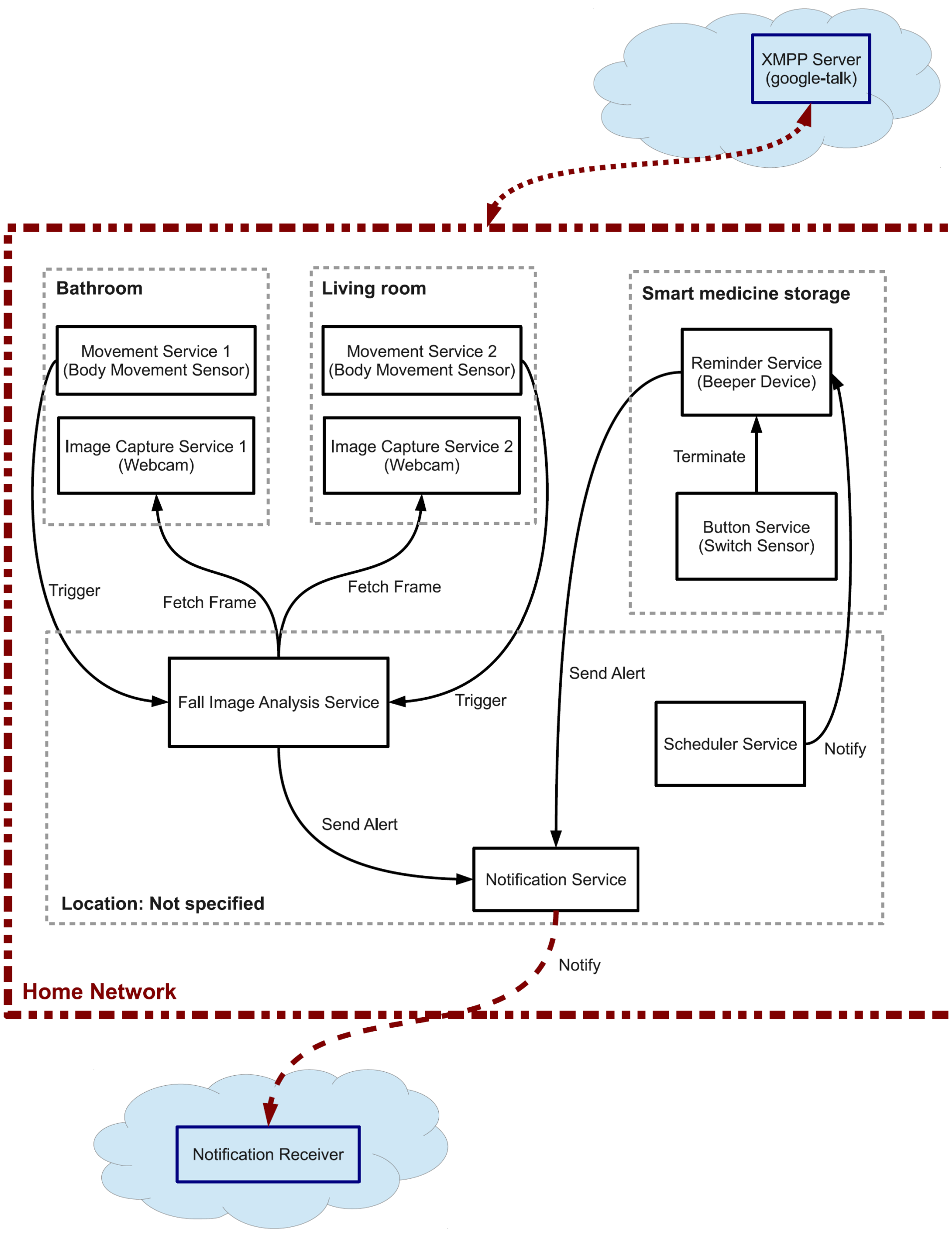

FIGURE 1. The Services and the messages of the telecare application

4.3. **Message formats.** In the following paragraphs, we describe the message formats used in the services. They are the service-level protocol for interoperability. Some of the message are trivial and can be designed without a real format. For example,

- The image capture service is passive. It responds to an arbitrary message with the captured binary image data (via the dedicated TCP channel).
- The movement service only sends a message to indicate the “movement detected” situation.
- The scheduler service sends a message to the reminder service. The message is specified when registering a timed event (the registration message is discussed later).

- The button service only sends a message to indicate the "lid-open detected" situation.

The notification messages are required when we need to associate a service to a caretaker. As mentioned, the notification service handles the subscription and the event dispatching. The related message formats are listed below:

- Subscription related messages (issued by event receiver)
  - `( 'subscribe', <publisher's full JID>, <event type> )`
  - `( 'unsubscribe', <publisher's full JID>, <event type> )`
- Event dispatching related messages (issued by event sender)
  - `( 'register', <event type> )`
  - `( 'unregister', <event type> )`
  - `( 'publish', <event type>, <event data> )`

In the above messages, the issuer's JID is not necessary because it is specified as the source JID in the XMPP message header. An `<event type>` is a string that represents the name of the event. It is used to reflect the generic situation. The occurrence-dependent information should be specified in the `<event data>`.

The scheduler service generates the timed event. It is also implemented by the publisher-subscriber pattern. The registration message format is designed as

- `( (<start time>, <interval>, <stop time>), <message> )`
  - `<start time>` (inclusive): string `<YYYY-MM-DD hh:mm:ss>`. If `<start time>` is None, it indicates the current time.
  - `<interval>:` string `<dd:hh:mm:ss>` where dd represents the number of days. If `<interval>` is None, it indicates that the message will be sent once.
  - `<stop time>` (exclusive): string `<YYYY-MM-DD hh:mm:ss>`. If `<stop time>` is None, it indicates that there is no limit on the number of times the message should be sent.
  - `<message>:` the string that will be sent to the issuer every time the timed event occurs.
- `( 'cancel', <key>)`

Upon a successful timed event registration, the corresponding `<key>` will be sent to the issuer. The issuer can use the key to unregister the timed event. The design of the scheduler message format supports periodic timed events because medicine-taking may be periodic.

4.4. **Implementation.** In this example, we need to integrate pure-software components and sensors. To make a sensor accessible in the service framework, we connect the physical device to a computer and write a wrapper service that communicates with the sensor to obtain the readings. The connection between a sensor and the computer would be wired or wireless, depending on the deployment requirements. Here we assume the connections are wired and it is sufficient for demonstration purpose. In the implementation, we connect the sensors to the Arduino Mega 2560 board. The program uploaded to the Arduino board translates the sensor readings to a value recognized by the corresponding wrapper service. The wrapper communicates with the Arduino board through the USB (emulated RS232) port. The software services are implemented with the Python SleekXMPP [21] library. The implemented service components are list below:

- Movement service (wrapper): detects abrupt body movement.
- Image capture service (wrapper): captures an image ondemand.
- Fall image analysis service (pure-software): fetches an image from the image capture service analyze it, and sends a notification if fall is detected.

- Scheduler service (pure-software): sends a message to the registered service at a specific time.
- Reminder service (pure-software): produces beeps on request and sends notification if timeout.
- Button service (wrapper): detects box lid open-close and sends messages to stop the reminder.
- Notification service (pure-software): receives an event and multi-casts it to the subscribers.

Due to the limit of the picture view we only show the configuration of the body movement sensor with the webcam. In Fig 2 the computer is a single board Raspberry Pi, a wireless network interface is attached directly to the USB port The webcam and the Ardiono (with the movement sensor) are connected to the computer via an USB hub. The movement service and the capture service run on the computer and join the telecare application by wireless LAN

5. **Discussions.** We described the design and the implementation of a telecare application from the core framework (Sec. 3) to the specific services (Sec. 4). There are some opinions learned from the process, and we will discuss them in the following paragraphs.

A concern is how to specify the feature of a service. There are standards for service description, such as the WSDL. We may follow the standards to define the feature. However, standard description syntax is sophisticated because it is designed for various kinds of interoperability. A prototyping project, like our example, is usually focused on the functionality and configuration. Since interoperability is not the major concern and there are no restrictions on XMPP feature strings, we may define simple string formats that are recognized by the project developers in the prototyping phase. When it is time to consider the interoperability, we can revise the program to process standard service descriptions without much trouble.

The notification service and the scheduler service are implemented by the publisher-subscriber pattern. We ecided to separate the two mechanisms instead of reusing the common event handling procedures. To be more specific, the scheduler service may use the notification service to publish the event. It is slightly more flexible than the current design. However, the dependency between the two services increases the complexity of the system. For example, the reminder service registers a timed event with the scheduler service. The scheduler service registers the event to the notification service and redirects the reminder service to subscribe it. This induces the indirect dependencies among the services.

6. **Conclusions.** Telecare technology is used to help less able people. In this paper, we proposed a software framework for prototyping telecare oriented applications. Although a production-grade telecare application is typically compact and customized, there are cases that we need a distributed environment to deploy various kinds of low-cost hardware/software sensors and services. The service oriented architecture (SOA) is suitable for software applications that integrate distributed components. The XMPP specifications are open and can be used to develop efficient message exchanging programs. We combine the two technologies to form our framework. The core design is the fundamental service. It is responsible for connecting to the XMPP server and discovering the necessary services. The detailed procedure and the service description concerns were also discussed.

A telecare scenario was chosen to demonstrate the effectiveness of the framework. We showed the development process of the application. Based on the requirements, the necessary services were identified and the deployment structure was designed. The message formats were then specified to support the interactions among the services, and finally the services were implemented as pure-software components

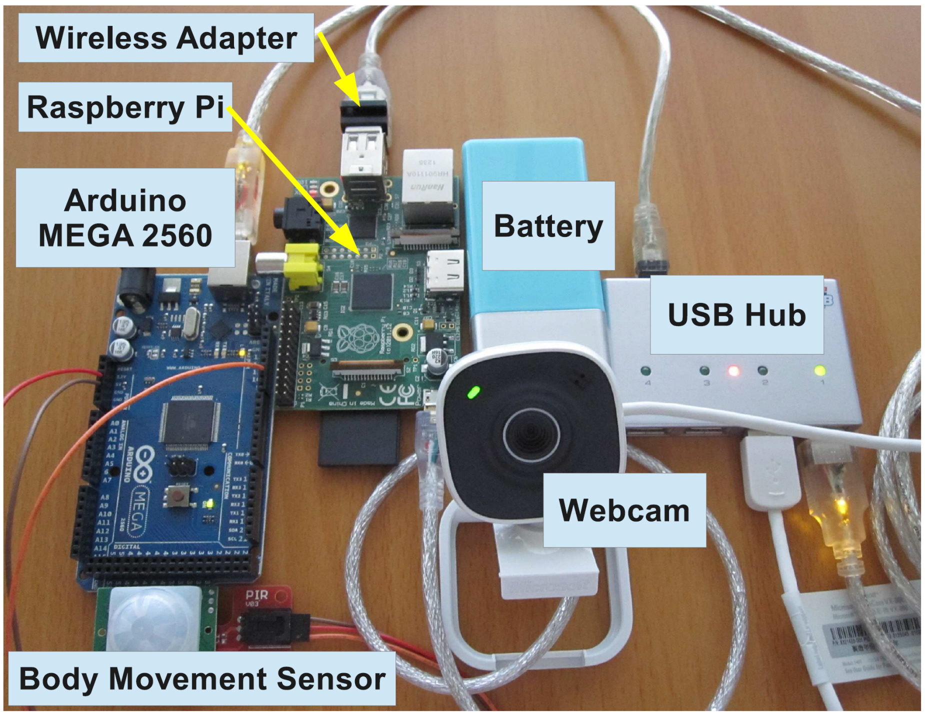

FIGURE 2. Physical configuration of image capture service and movement service

and hardware wrappers. The included services are: the fall detection function that is composed by a movement service, an image capture service and a fall image analysis service; the reminder function that integrates a scheduler service a reminder service and a button service; and the notification service that handles event dispatching to the caretakers The flexibility is shown in two aspects: the service implementation is protocol centric; and the important XMPP network entity the server(s) could a thirdparty service such as Google-talk

Acknowledgement. This work was partially supported by the NSC Taiwan, under Grants NSC 99-2221-E-032-050 and NSC 101-2221-E-032-052 The hardware devices used in the projects were partially supported by Tamkang University Taiwan, under grant of the subsidiary project for Technology and Society (year 2011).

REFERENCES

[1] J. M. Kang, T. Yoo, and H. C. Kim, A wristworn integrated health monitoring instrument with a tele-reporting device for telemedicine and telecare, *IEEE Trans. Instrumentation and Measurement*, vol. 55, no. 5, pp. 1655-1661, 2006.

[2] J. Barlow, D. Singh, S. Bayer, and R. Curry, A systematic review of the benefits of home telecare for frail elderly people and those with longterm conditions, *Journal of Telemedicine and Telecare*, vol. 13, no. 4, pp. 172-179, 2007.

[3] H. Takahashi, K. Yamanaka, S. Izumi, Y. Tokairin, T. Suganuma, and N. Shiratori, Gentle supervisory system based on integration of environmental information and social knowledge, *International Journal of Pervasive Computing and Communications*, vol. 6, no. 2, pp.229-247, 2010.

[4] S. J. Hsu, H. H. Wu, S. W. Chen, T. C. Liu, W. T. Huang, Y. J. Chang, C. H. Chen, and Y. Y. Chen, Development of telemedicine and telecare over wireless sensor network, *Proc. of The 2nd International Conference on Multimedia and Ubiquitous Engineering*, pp.597-604, 2008.

[5] D. Berian, V. Gomoi, and V. Topac, A hybrid solution for a telecare system server, *Proc. of The 6th IEEE International Symposium on Applied Computational Intelligence and Informatics*, pp. 589-592, 2011.

[6] C. H. Chen, W. T. Huang, Y. Y. Chen, and Y. J. Chang, An integrated service model for telecare system, *Proc. of The 3rd IEEE Asia-Pacific Services Computing Conference*, pp. 712-717, 2008.

[7] H. C. Huang, W. C. Fang, and W. H. Lai, Secure medical information exchange with reversible data hiding, *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 1424-1427, 2012.

[8] H. C. Huang, Y. H. Chen, and A. Abraham, Optimized watermarking using swarm-based bacterial foraging, *Journal of Information Hiding and Multimedia Signal Processing,* vol. 1, no. 1, pp. 51-58, 2010.

[9] R. Varadan, K Channabasavaiah, S. Simpson, K. Holley, and A. Allam, Increasing business flexibility and SOA adoption through effective SOA governance, *Journal of IBM Systems Journal*, vol. 47, no. 3, pp. 473-488, 2008.

[10] R. Kyusakov, J. Eliasson, J. Delsing, J. V. Deventer, and J. Gustafsson, Integration of wireless sensor and actuator nodes with IT infrastructure using service-oriented architecture, *IEEE Trans. Industrial Informatics*, vol. 9, no. 1, pp. 43-51, 2013.

[11] C. L. Wu, C. F. Liao, and L. C. Fu, Service-oriented smartHome architecture based on OSGi and mobile-agent technology, *IEEE Trans. Systems Man and Cybernetics Part C: Applications and Reviews*, vol. 37, no. 2, pp. 193-205, 2007.

[12] N. Bieberstein, S. Bose, L. Walker, and A. Lynch, Impact of service-oriented architecture on enterprise systems organizational structures and individuals, *Journal of IBM Systems Journal*, vol. 44, no. 4, pp. 691-708, 2005.

[13] M. P. Papazoglou, and W. J. Heuvel, Service oriented architectures: approaches technologies and research issues, *Journal of The VLDB Journal*, vol. 16, no. 3, pp. 389-415, 2007.

[14] P. Saint-Andre, XMPP: lessons learned from ten years of XML messaging, *Journal of IEEE Communications Magazine*, vol. 47, no. 4, pp. 92-96, 2009.

[15] F. C. Chang, and D. K. Chen, The design of an XMPP-based service integration scheme, *Proc. of The 7th International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 33-36, 2011.

[16] P. Saint-Andre, K. Smith, and R. Troncon, *XMPP: The Definitive Guide,* O'Reilly, Sebastopol, US, 2009.

[17] I. Sommerville, *Software Engineering*, Addison-Wesley, Boston, US, 2011.

[18] Object Management Group, *Common object request broker architecture*, v3.1 Technical report, available at http://wwwomgorg/spec/CORBA/31/.

[19] D. Booth, and C. Kevin Liu, *Web services description language (WSDL) version 2. part : Primer*, W3C, available at http://www.w3org/TR/2007/RECwsdl20-primer2007626.

[20] Y. Rao, and L. Chen, A survey of video enhancement techniques, *Journal of Information Hiding and Multimedia Signal Processing*, vol. 3, no. 1, pp. 71-99, 2012.

[21] N. Fritz, *SleekXMPP*, available at http://sleekxmpp.com/