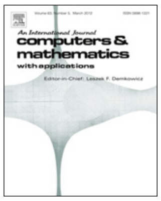
Computers and Mathematics with Applications 64 (2012) 1324–1332

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa



A healthcare system as a service in the context of vital signs: Proposing a framework for realizing a model

Tae-Woong Kim, Hee-Cheol Kim*

Department of Computer Engineering/UHRC, Inje University, Gimhae, Gyeong-Nam, Republic of Korea

ARTICLE INFO

Keywords:
Cloud computing
Software as a service (SaaS)
Service-oriented architecture (SOA)
Ubiquitous healthcare
Vital signs

ABSTRACT

In an age of cloud computing, users do not access a particular system, but a service. In a more centralized and seamless environment, it becomes important to view software as a service, or "SaaS". In particular, ubiquitous healthcare urgently needs to reflect the view of the healthcare system as a service, "HaaS", for the purposes of health promotion, disease prevention, and general well-being. Overcoming the current limitations of medical technologies, medical practices, and business, HaaS requires a more sustainable environment, including addressing issues of interoperability, service reuse, maintenance, and integration. This paper suggests a framework for realizing a HaaS model, focusing on vital signs-oriented systems. The proposed framework consists of four distinct modules: three modules for receiving, transforming, and analyzing vital signs and the fourth module, called a service-oriented architecture (SOA) component module, enabling access to medical services. We believe that the framework promises dramatic reductions in software development costs on the one hand and lays a foundation for users to access a rich and seamless service in a more convenient way on the other. This paper describes a framework for the HaaS model and presents an example service that we implemented, a calorie-tracking service, based on it.

1. Introduction

As more attention is paid to cloud computing, software as a service (SaaS [1]) is becoming a key delivery model. By defining and viewing software as a service, one can avoid redundant development of similar types of software and share software that already exists via the cloud net. Users are no longer device dependent (or system dependent), but instead are more service oriented [2]. Together with virtual-server technology, SaaS is becoming vital to success in this ubiquitous cloud environment.

Particularly with the coming age of ubiquitous computing (u-computing) [3], demand for u-healthcare services is increasing dramatically [4]. One advantage of u-healthcare practices is well known: users can check their health status at any time, anywhere, and with any device, Vital signs, such as respiration, blood pressure, pulse, and electrocardiography (ECG), are especially important for these purposes because these signals are indicators of one's health status. When they are acquired and analyzed at any time and anywhere, this will eventually help to promote health and prevent disease. A great barrier arises, however, in that software for evaluating health states is dependent on a particular device, terminal, or platform, which causes a lack of interoperability among different healthcare systems in terms of data format.

The model that we propose, healthcare systems as a service (HaaS), is a SaaS model wherein healthcare systems are defined as services in order to secure interoperability. By this model, a device or terminal is used to measure vital signs and

* Corresponding author. Tel.: +82 55 320 3720; fax: +82 55 322 3107.
E-mail address: heeki@inje.ac.kr (H.-C. Kim).

0898-1221/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2012.03.076

transmits them to a virtual cloud environment. One can then check the patient's health status at any time and anywhere by accessing the virtual server. This is a main way in which HaaS is like SaaS.

This paper aims to provide a framework to support the HaaS model. The proposed framework consists of four distinct modules: a module to receive vital sign data using standardized messaging methods; a module to transform these data into a standardized schema; a module to evaluate the health state using biosignal data; and a service-oriented architecture (SOA) component module that allows users to access medical services with standardized messaging methods. In particular, developing the third module, we used Object Constraint Language (OCL), a platform-independent specification language, by which it accesses vital signs and evaluate a person's state of health, having a structure distinct from that of existing components in enterprises.

If the proposed framework for supporting the HaaS model is adopted, one can expect development costs to be reduced substantially, as there is little need to develop software that is embedded in a particular terminal or independent device. Service users will also benefit from the added convenience and mobility. Furthermore, as vital signs are accumulated every day in the cloud environment in a standardized and easy way, we have improved chances for understanding and caring for our health from a historical perspective.

Section 2 describes the background, including the SOA component structure in general, the OCL and vital signs-based healthcare systems. Section 3 proposes the framework supporting the HaaS model. Section 4 presents a calorie-tracking service that we implemented using this system, followed by our conclusions.

2. Background

2.1. SOA component structures

Service-oriented architecture (SOA) is a software architecture that enables interoperability, defining ways to integrate widely disparate applications using multiple implementation platforms. It allows developers to compose and reuse components, called services, over the net. Services communicate with one another, coordinating activity among three or more services in standardized ways [5,6]. The potential benefits of SOA include service reuse, improved integration, leveraged legacy investment, and best-of-breed integration [7]. Using SOA technology, one can eventually develop new business applications rapidly and easily by combining different services [8].

For healthcare, standardization organizations such as Health Level Seven (HL7), Object Management Group (OMG) [9], and Health Informatics Service Architecture [10] work to produce healthcare-specific service architectures and web services. SOA-based healthcare systems have been also developed, including disease-notification [11] and health-monitoring [12] systems. Furthermore, studies have examined models and design considerations for service-based application development and integration [13,14].

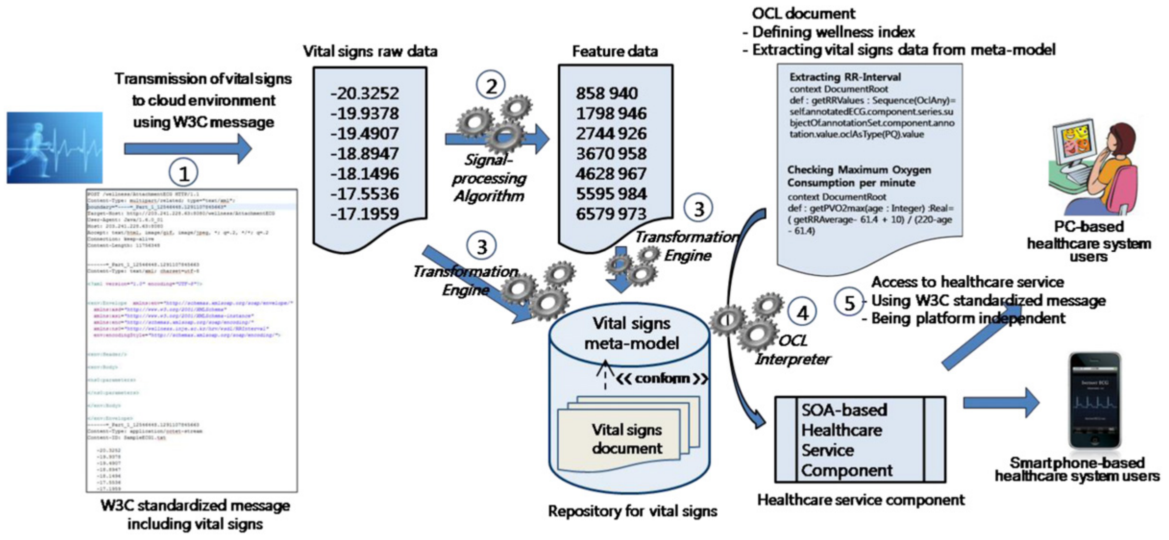
2.2. Object Constraint Language

OCL [15], which was standardized by OMG, is a specification language that describes structural constraints over models expressed in Unified Modeling Language (UML) [16]. It expresses constraints about specific elements in UML diagrams using invariants and conditions [17], and defines elements that cannot be expressed in existing models using functions. Because of these advantages, OCL is used for problem-solving tasks such as model verification [18,19] and test-case generation [20]. More specifically, OCL is a mathematical specification language that defines the state of an object or specifies constraints that the object must have. OCL is also used to apply an object to a data model described on the basis of a metamodel and to detect its state.

For example, OCL has been used in a specific domain where the structure of a system detects particular features of the Java source codes of an application program; the codes are then transformed into a metamodel-based abstract syntax tree [21]. It considers potential problems in source codes, called "Bad Smells in Code" [22,23]. Features of interest include source code caused by bad software design and bad programming practices, such as the overuse of switch-case statements and too many parameters, complex message chains, unnecessary class inheritance, and complex class hierarchy. The system can detect specific features of a given document, by applying OCL to a metamodel-based document model. Likewise, OCL can be used to detect features of an individual's health in a healthcare system from vital signs data on the basis of a metamodel. Features of the state of health include the body mass index, maximum oxygen consumption per minute ($VO_{2\max}$), calorie consumption, health-risk appraisal, stress index, and fitness index.

2.3. Vital signs-based healthcare systems

At present, some vendors provide healthcare services that make use of vital signs to interpret the state of personal health and detect and prevent diseases. Polar and Adidas have collaborated to develop an integrated training system [24]. Honeywell makes a patient-monitoring system using the HomMed LifeStream platform, which dispatches nurses to the patient's home when a health problem is detected [25]. Biocom and Laxtha developed and market a packaged health-monitoring system that gathers and analyzes personal health-related information [26,27]. Additionally, information


Transmission of vital signs to cloud environment using W3C message
1
W3C standardized message including vital signs
Vital signs raw data
-20.3252
-19.9378
-19.4907
-18.8947
-18.1496
-17.5536
-17.1959
2
Signal-processing Algorithm
Feature data
858 940
1798 946
2744 926
3670 958
4628 967
5595 984
6579 973
3
Transformation Engine
3
Transformation Engine
Vital signs meta-model
<< conform >>
Vital signs document
Repository for vital signs
OCL document
- Defining wellness index
- Extracting vital signs data from meta-model
4
OCL Interpreter
SOA-based Healthcare Service Component
Healthcare service component
5
Access to healthcare service
- Using W3C standardized message
- Being platform independent
PC-based healthcare system users
Smartphone-based healthcare system users

```
POST /wellness/AttachmentECG HTTP/1.1
Content-Type: multipart/related; type="text/xml";
boundary="----=_Part_1_12546448.1291107845663"
Target-Host: http://203.241.228.43:8080/wellness/AttachmentECG
User-Agent: Java/1.6.0_01
Host: 203.241.228.43:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 11756348

------=_Part_1_12546448.1291107845663
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ns0="http://wellness.inje.ac.kr/hrv/wsdl/RRInterval"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<env:Header/>
<env:Body>
<ns0:parameters>
</ns0:parameters>
</env:Body>
</env:Envelope>
------=_Part_1_12546448.1291107845663
Content-Type: application/octet-stream
Content-ID: SampleECG1.txt

  -20.3252
  -19.9378
  -19.4907
  -18.8947
  -18.1496
  -17.5536
  -17.1959
```

```
Extracting RR-Interval
context DocumentRoot
def : getRRValues : Sequence(OclAny)=
self.annotatedECG.component.series.su
bjectOf.annotationSet.component.anno
tation.value.oclAsType(PQ).value

Checking Maximum Oxygen
Consumption per minute
context DocumentRoot
def : getPVO2max(age : Integer) :Real=
( getRRAverage- 61.4 + 10) / (220-age
- 61.4)
```

Fig. 1. A framework for HaaS.

technology giants such as IBM, Philips, and LG produce special terminals or provide applications for analyzing vital signs obtained from various devices and support online counseling, such as for health-risk management.

However, the systems mentioned here are all platform dependent. Each device or application needs a particular platform, and uses a distinct format. Thus, it is very difficult to build an integrated solution system or service. Given each vendor's characteristics, it is also difficult to construct an integrated database. Furthermore, in terms of ways of diagnosing health status, developers must use different programming languages to process and analyze biosignal data and to define and evaluate the state of health. Given all these problems, biosignal data from one medical device cannot be used in another system. Additionally, methods of diagnosis used in one system cannot be used in another. That is, interoperability between different systems is not guaranteed. Thus, we need a standardized healthcare system to manage biosignals data from different devices and systems.

3. A framework for HaaS

3.1. *A framework*

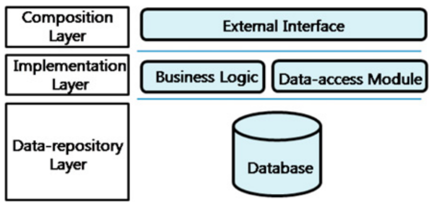
In an environment of cloud computing, healthcare-system users ideally will be able to access services at any time, anywhere, and using any device. Fig. 1 shows a summary of the proposed framework for achieving the rationale of HaaS in the context of a vital signs-oriented healthcare system.

The framework that we suggest has the following characteristics:

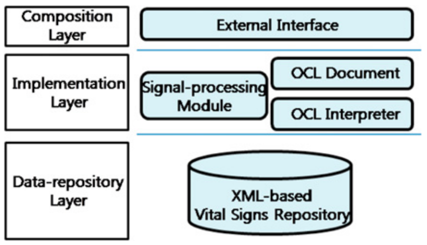
① Biosignals obtained are transmitted to the server using W3C standardized messaging [28]. Here, note that "server" implies a personalized repository in a cloud environment.
② Information is extracted from the values related to features that are required to evaluate the health state from the biosignals raw data. For this, appropriate algorithms are applied.
③ Both the raw data and the extracted feature data are represented as XML documents, based on HL7, and saved in a personalized repository.
④ A wellness index is generated, defined in OCL and in a SOA-based healthcare service component, including an OCL evaluator.
⑤ The healthcare component enables users to access services with various devices using the standardized W3C message protocol [28].

3.2. *The structure of the healthcare service component*

Generally, the SOA components in application programs use business logic. Consequently, such SOA components are unsuitable for vital signs-based healthcare systems because the SOA components of healthcare systems must include algorithm modules rather than business logic. Such algorithm modules process and analyze the vital signs data. Moreover, the data types used in enterprise application programs are typically quite different from those in biosignal-based healthcare applications.



(a) Internal structure of the SOA components in business application programs.



(b) Internal structure of the SOA components in vital signs-based healthcare systems.

Fig. 2. SOA structure of a vital signs-based healthcare service system.

The data processed in enterprise application programs are stored in many rows. This makes it more efficient to store and access the data in the DB. In contrast, vital signs-based healthcare systems deal mostly with data represented in columns. This implies that file structures written as blocks are much better for representing the data than DBs are. Thus, it is better to design the SOA component to include modules that store and access data in a file, but not in a DB.

Whereas a DB contains semantic information about the data that is accessed using a query language, a file involves more difficult semantics. Moreover, there is no consistent way to access the data in a file. Because each healthcare system uses different file formats, it is difficult to develop a general-purpose SOA component that will work in various client environments. Importantly, however, this problem can be resolved by expressing the data in the form of a metamodel-based extensible markup language (XML) document.

The SOA components of healthcare systems should be designed to meet the following requirements:

- Vital signs data should be represented using standardized structures.
- Access to vital signs data should be easy and fast, and the methods should be consistent.
- Modules for checking the state information of vital signs data should be included.
- Algorithm modules for processing vital signs data should be included.
- A general, standardized method should be designed and implemented.

Here, we propose an SOA component for healthcare systems that satisfies these requirements as follows (see Fig. 2(b)):

- It expresses vital signs data using a standard metamodel based on HL7.
- It expresses the vital signs data using standard XML.
- It retrieves vital signs data and the health status using OCL.

The advantage of XML files accepting the HL7 standard and a metamodel is that this approach enables interoperability and supports a consistent mode of data access. Earlier, we mentioned that a strength of OCL is apparent when a system applies an object to a metamodel-based data model and detects its state. Because biosignal data are stored in standardized XML, OCL can be used to access the data in an XML document and to implement a module for evaluating the state of the data. Thus, it is natural to use OCL as an element of the SOA component.

3.3. Representation of vital signs: an example of ECG and its related HRV data

As shown in Fig. 3, the way to represent vital signs in the SOA component structure is to use a standardized, metamodel-based representation. As an example, we use ECG data for checking the health status.

ECG waves are generally represented graphically (Fig. 3(a)). The raw wave data are obtained from medical devices in file form (Fig. 3(b)). It is then useful to extract the heart-rate variability (HRV) from the ECG measured over a defined period of time. HRV is a physiological phenomenon in which the interval between heartbeats varies. An ECG data set consists of P, Q, R, S, and T waves [29], with the R peaks having the greatest magnitude. As a measure of cycle-length variability, HRV is a time series of intervals between successive R peaks (RR intervals) in the ECG. Fig. 3(c) shows the format of a file expressing the values of these RR intervals, acquired using a signal-processing algorithm. Generally, HRV is useful for evaluating autonomic nervous system (ANS) function, whereas the ECG is better for diagnosing various heart-related diseases. Technically, the data in Fig. 3(b) and (c) are transformed into the XML document, shown in Fig. 3(d), with the help of a transformation engine. The final data obtained by this process are stored in an XML-based vital signs repository in the data repository layer of the SOA component proposed here.

3.4. Access to vital signs data and inference of the state of health

Although it is useful to store vital signs data in a standardized manner in a healthcare system, it is more important to access and retrieve the data and to evaluate the state of health. It is a common practice in software development to store data in a DB, including vital signs data, and to access the data via a related query. In this case, the state of health is generally

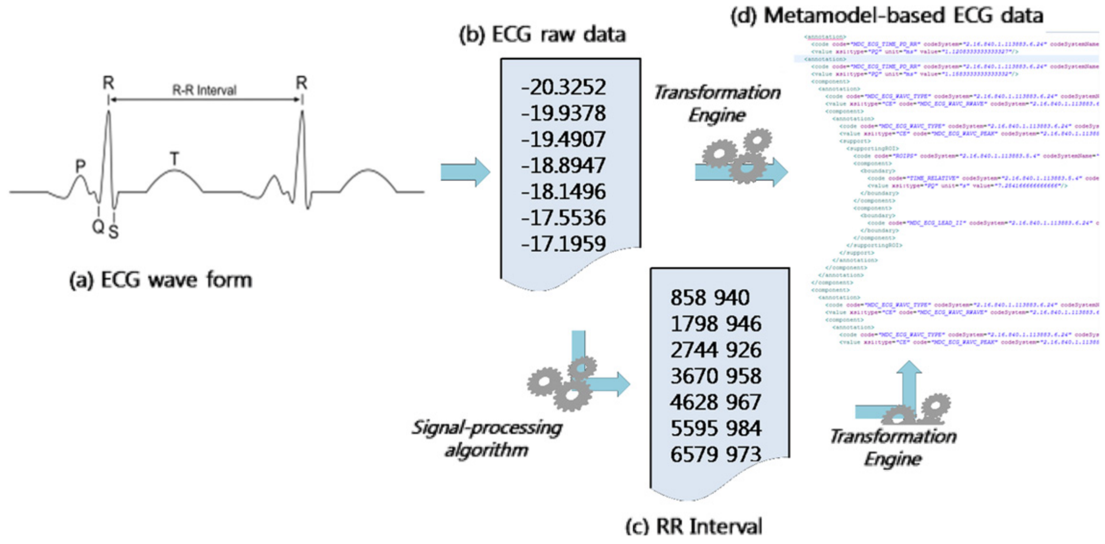


Fig. 3. Transformation process for metamodel-based ECG data.

evaluated using programming languages. However, this strategy has difficulty coping with updates to the modules that check the state of health, which changes constantly, and a DB is inadequate for processing vital signs. To overcome this problem, we take a new approach in which the system retrieves the vital signs data and checks the health state using OCL as part of the implementation layer shown in Fig. 2(b).

Using OCL, a mathematical specification language, it is much easier to add to and update the module checking the state of health. OCL has better readability than other programming languages. Developers can conveniently insert and delete source code, and additional tasks, such as recompilation, are not needed.

4. Framework implementation: an example

A well-known index of health is $VO_{2\max}$, the maximum oxygen consumption per minute, which is useful for estimating the calories burnt during exercise [30]. Interestingly, $VO_{2\max}$ can be approximated from HRV in a meaningful manner. Eqs. (1)–(3) demonstrate the process used to convert HRV into $VO_{2\max}$, and $VO_{2\max}$ into calories burned [30]. Additionally, although its use is limited, one can evaluate the ANS function using the calories calculated and the ratio of low frequency to high frequency HRV. In this paper, we implement a component that estimates calories burned using the HRV extracted from the ECG as an example.

$$\%VO_{2\max} = \frac{\mathrm{HRV} - \mathrm{HRV}_R + 10}{\mathrm{HRV}_M - \mathrm{HRV}_R} \tag{1}$$

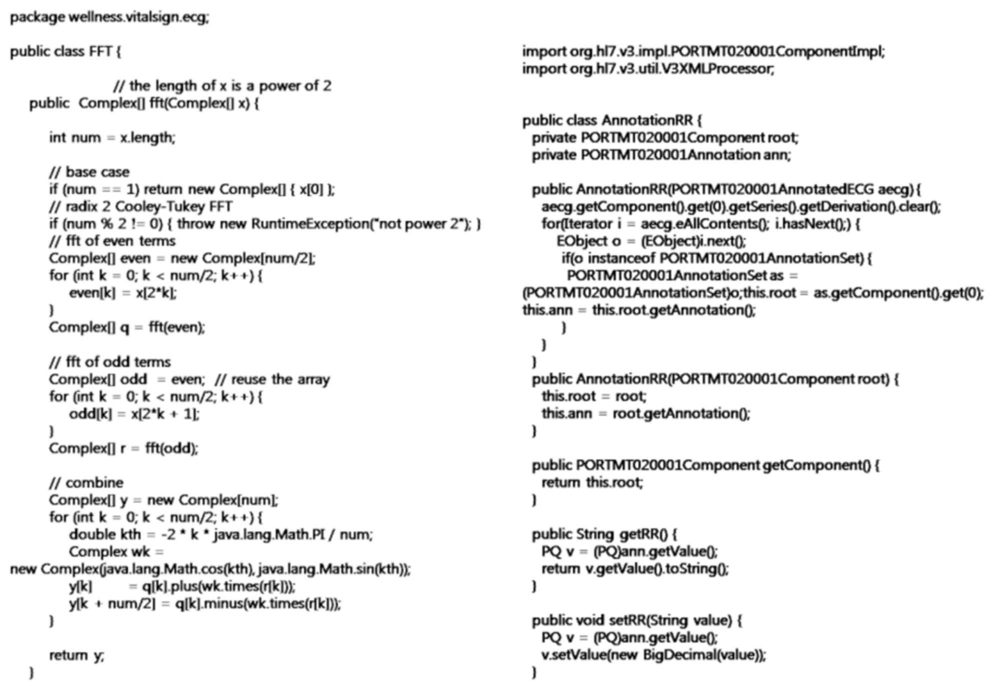
$$\text{Male: CAL} = \frac{(3.83 \times t + 13) \times M \times \%VO_{2\max} \times t}{0.2} \tag{2}$$

$$\text{Female: CAL} = \frac{(3.83 \times t + 13)/8 \times M \times \%VO_{2\max} \times t}{0.2} \tag{3}$$

$\%VO_{2\max}$	= Maximum oxygen consumption per minute
HRV	= Average HRV
HRV_R	= Average HRV according to age group
HRV_M	= Maximum HRV according to age group

4.1. Transformation of raw ECG data into the annotated ECG format

When raw ECG data obtained from a device are sent to a server, they are converted into a SOAP message, enabling their use in various platforms (i.e., the data are then platform independent). Because SOAP messages are text, they can be created by any language that can handle text strings. From the raw ECG data sent to the server, the system creates an aECG-based XML document [31] that includes the ECG signal data processed using an HRV-extracting algorithm (Fig. 4(a)) and a transformation engine (Fig. 4(b)). Here, the HL7 aECG is a medical record data format for storing/retrieving the ECG



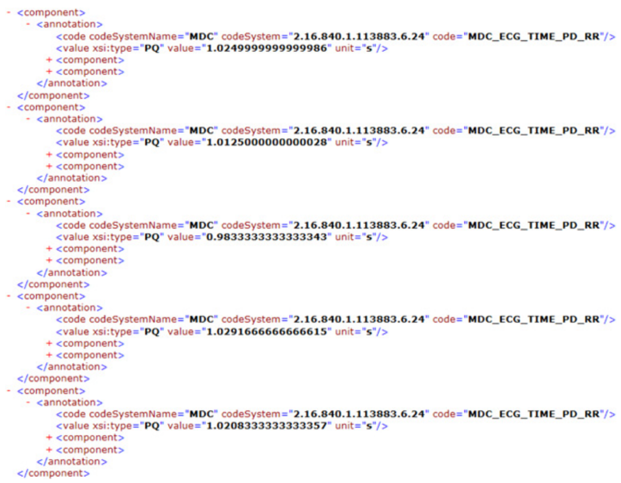
```
package wellness.vitalsign.ecg;

public class FFT {

                // the length of x is a power of 2
   public  Complex[] fft(Complex[] x) {

      int num = x.length;

      // base case
      if (num == 1) return new Complex[] { x[0] };
      // radix 2 Cooley-Tukey FFT
      if (num % 2 != 0) { throw new RuntimeException("not power 2"); }
      // fft of even terms
      Complex[] even = new Complex[num/2];
      for (int k = 0; k < num/2; k++) {
         even[k] = x[2*k];
      }
      Complex[] q = fft(even);

      // fft of odd terms
      Complex[] odd  = even;  // reuse the array
      for (int k = 0; k < num/2; k++) {
         odd[k] = x[2*k + 1];
      }
      Complex[] r = fft(odd);

      // combine
      Complex[] y = new Complex[num];
      for (int k = 0; k < num/2; k++) {
         double kth = -2 * k * java.lang.Math.PI / num;
         Complex wk =
new Complex(java.lang.Math.cos(kth), java.lang.Math.sin(kth));
         y[k]       = q[k].plus(wk.times(r[k]));
         y[k + num/2] = q[k].minus(wk.times(r[k]));
      }

      return y;
   }
```

(a) Signal-processing algorithm for extracting HRV.

```
import org.hl7.v3.impl.PORTMT020001ComponentImpl;
import org.hl7.v3.util.V3XMLProcessor;


public class AnnotationRR {
  private PORTMT020001Component root;
  private PORTMT020001Annotation ann;

  public AnnotationRR(PORTMT020001AnnotatedECG aecg) {
    aecg.getComponent().get(0).getSeries().getDerivation().clear();
    for(Iterator i = aecg.eAllContents(); i.hasNext();) {
       EObject o = (EObject)i.next();
        if(o instanceof PORTMT020001AnnotationSet) {
         PORTMT020001AnnotationSet as =
(PORTMT020001AnnotationSet)o;this.root = as.getComponent().get(0);
this.ann = this.root.getAnnotation();
        }
    }
  }
  public AnnotationRR(PORTMT020001Component root) {
    this.root = root;
    this.ann = root.getAnnotation();
  }

  public PORTMT020001Component getComponent() {
    return this.root;
  }

  public String getRR() {
    PQ v = (PQ)ann.getValue();
    return v.getValue().toString();
  }

  public void setRR(String value) {
    PQ v = (PQ)ann.getValue();
    v.setValue(new BigDecimal(value));
  }
```

(b) Transformation engine for converting ECG raw data to an aECG-based XML document.

Fig. 4. Source codes for an HRV-extraction algorithm and transformation engine.



```
- <component>
  - <annotation>
      <code codeSystemName="MDC" codeSystem="2.16.840.1.113883.6.24" code="MDC_ECG_TIME_PD_RR"/>
      <value xsi:type="PQ" value="1.0249999999999986" unit="s"/>
    + <component>
    + <component>
    </annotation>
  </component>
- <component>
  - <annotation>
      <code codeSystemName="MDC" codeSystem="2.16.840.1.113883.6.24" code="MDC_ECG_TIME_PD_RR"/>
      <value xsi:type="PQ" value="1.0125000000000028" unit="s"/>
    + <component>
    + <component>
    </annotation>
  </component>
- <component>
  - <annotation>
      <code codeSystemName="MDC" codeSystem="2.16.840.1.113883.6.24" code="MDC_ECG_TIME_PD_RR"/>
      <value xsi:type="PQ" value="0.9833333333333343" unit="s"/>
    + <component>
    + <component>
    </annotation>
  </component>
- <component>
  - <annotation>
      <code codeSystemName="MDC" codeSystem="2.16.840.1.113883.6.24" code="MDC_ECG_TIME_PD_RR"/>
      <value xsi:type="PQ" value="1.0291666666666615" unit="s"/>
    + <component>
    + <component>
    </annotation>
  </component>
- <component>
  - <annotation>
      <code codeSystemName="MDC" codeSystem="2.16.840.1.113883.6.24" code="MDC_ECG_TIME_PD_RR"/>
      <value xsi:type="PQ" value="1.0208333333333357" unit="s"/>
    + <component>
    + <component>
    </annotation>
  </component>
```

Fig. 5. An aECG XML document where RR interval values are expressed.

data for a patient. Like other HL7 formats, it is XML based. The HL7 aECG standard was created by the Regulated Clinical Research Information Management (RCRIM) working group of HL7 in response to the digital electrocardiogram initiative of the US Food and Drug Administration, introduced in November 2001 [32].

4.2. OCL code and the service component for measuring calorie consumption

The role of OCL code is not confined to accessing and detecting the data in an XML document but also extends to evaluating the state of the object model. As mentioned in Section 2.2, if the raw data are expressed and included in a metamodel-based XML document (Fig. 5), the system can check the state of the given object. Adopting this, we applied it to vital signs data, specifically ECG data. In fact, the state of the ECG data reflects the state of health. Thus, it is possible to check the state of health from the ECG data in a metamodel-based XML document to some extent.

Table 1
Functions in the OCL code.

Function name	Description
getRRValues	Returns a set of *R*-peak values in an aECG XML document
getBPM	Returns the average heart rate using the values of getRRValues
getPVO$_2$Max	Returns the value of the maximum oxygen consumption per minute
getCalorie	Returns the calories consumed by age, weight, gender, measurement time, and VO$_{2max}$.

The following OCL code handles the process used to calculate the calories burned from the HRV data. The code defines several functions (defined by 'def:') required to measure calorie consumption using OCL grammar, and Table 1 accounts for them.

```
context DocumentRoot
def: getRRValues: Sequence(OclAny) =
self.annotatedECG.component.series.subjectOf.annotationSet.component.annotation.value.oclAsType(PQ).value
def: getRRSize: Integer = getRRValues → size()
def: getHRVm(age: Integer): Integer = 220 − age
def: getEvaluateTime: Real = getRate * getRawDataSize
def: getBPM: Real = getRRValues → iterate(value: Real; answer: Real = 0 | answer + (60/value))/(getRRSize)
def: getPVO2max(age: Integer): Real = (getBPM − 61.4 + 10)/(getHRVm(age) − 61.4)
def: getCalorie(age: Integer, weight: Integer, sex: Integer): Real =
if sex = 0 then
    ((3.83 * (getEvaluateTime/60) + 13) * weight * getPVO2max(age) * (getEvaluateTime/60))/0.2
else
    ((3.83 * (getEvaluateTime/60) + 13 )/8 * weight * getPVO2max(age) * (getEvaluateTime/60))/0.2
endif
```

The approach proposed and tested here has the advantages of easier maintenance and increased readability because it does not use algorithms that depend on programming languages but rather uses OCL code independent of the source code.

Integrated with the OCL code above, the Java source code below indicates the method of a component that provides calories burned.

```
public class ECGService implements IECGService {
    public Double getCalorie(String id, String aECGname, int age, int weight, int sex) {
        Integer _age = new Integer(age);
        Integer _weight = new Integer(weight);
        Integer _sex = new Integer(sex);
        OCLEvaluator ocl = new OCLEvaluator();
        ocl.setXMLEnvironment(aECGPath + id + "/"+ aECGname +".xml");
        ocl.setOCLDefDocument(oclPath + "ECGService.ocl");
        Object result = ocl.checkQueryInRoot("getCalorie("+_age+","+_weight+","+_sex+")");
        Double calorie = (Double)result;
        return calorie;
        }
}
```

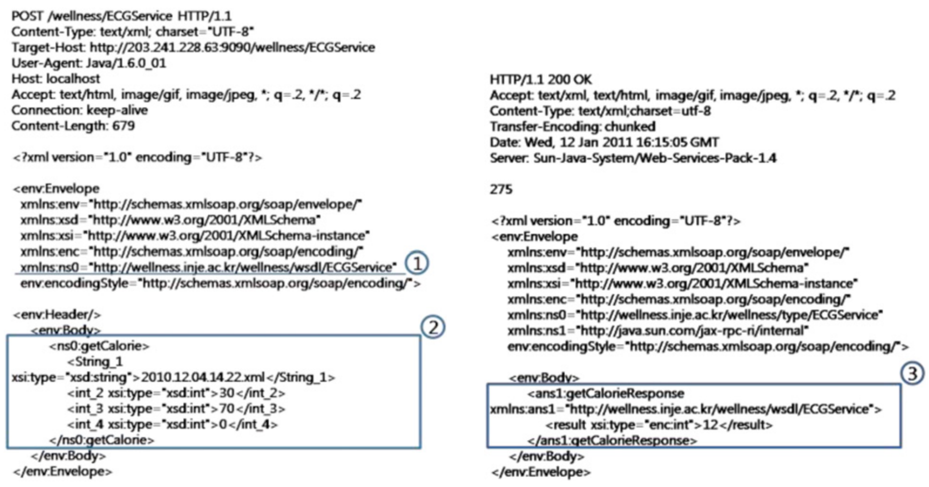
4.3. *Execution result: request messages and response messages*

This section shows the result obtained on accessing the SOA-based component that we developed for calorie estimation. The SOA component was designed to respond to a standard message. To request a service by accessing the SOA component on the server side, a client program writes a SOAP message (Fig. 6(a)) and sends it to the server via HTTP. Then, the SOA component creates a SOAP message (Fig. 6(b)) and returns it to the client.

Underline 1 in Fig. 4(a) defines the name space of the WSDL that is the ECG service, the SOA component implemented here. Box 2 defines the parameter value for requesting the 'getCalorie' method to get the value of calories burned, another ECG service. We can also see the values of aECG name, age, weight, and gender (male 0, female 1), in that order. Box 3 is a response message, and the value '12' was returned in the <result> element (i.e., 12 kcal were burned).

5. Conclusions

In a cloud computing world, users access services rather than systems, and software is not traditional software, but a service, SaaS. What is important is the structure by which various services are provided in a convenient, seamless, and low-cost way. HaaS, reflecting this, is a model that enables evaluation and management of one's health status at any time,



```
POST /wellness/ECGService HTTP/1.1
Content-Type: text/xml; charset="UTF-8"
Target-Host: http://203.241.228.63:9090/wellness/ECGService
User-Agent: Java/1.6.0_01
Host: localhost
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 679

<?xml version="1.0" encoding="UTF-8"?>

<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ns0="http://wellness.inje.ac.kr/wellness/wsdl/ECGService" ①
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<env:Header/>
    <env:Body>                                                        ②
        <ns0:getCalorie>
            <String_1
xsi:type="xsd:string">2010.12.04.14.22.xml</String_1>
            <int_2 xsi:type="xsd:int">30</int_2>
            <int_3 xsi:type="xsd:int">70</int_3>
            <int_4 xsi:type="xsd:int">0</int_4>
        </ns0:getCalorie>
    </env:Body>
</env:Envelope>
```

(a) Request SOAP message via HTTP.

```
HTTP/1.1 200 OK
Accept: text/xml, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Wed, 12 Jan 2011 16:15:05 GMT
Server: Sun-Java-System/Web-Services-Pack-1.4

275

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:ns0="http://wellness.inje.ac.kr/wellness/type/ECGService"
   xmlns:ns1="http://java.sun.com/jax-rpc-ri/internal"
   env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

   <env:Body>                                                         ③
       <ans1:getCalorieResponse
xmlns:ans1="http://wellness.inje.ac.kr/wellness/wsdl/ECGService">
           <result xsi:type="enc:int">12</result>
       </ans1:getCalorieResponse>
   </env:Body>
</env:Envelope>
```

(b) Response SOAP message via HTTP.

Fig. 6. Transformation process for metamodel-based ECG data.

anywhere, and using any device. This paper proposes a framework to realize the HaaS model, where vital signs-oriented healthcare systems are constructed within a virtual-server space.

The proposed framework is suitable for developing service components providing healthcare services. Because a client requests services using SOAP messages, services are also supported, independently of platforms, programming languages, and devices, including the use of personal computers and smart phones. By using such a framework, healthcare systems can be developed in a cloud environment. Consequently, users can access various services at any time and anywhere. Furthermore, healthcare-system developers can build software that follows the SaaS model. We also implemented an SOA-based calorie-tracking service as a fundamental element of a healthcare system.

At present, we have successfully completed testing of our approach using the Windows, Linux, Android, and iOS (for the iPhone and iPad) operating systems. Currently, we are working with doctors from the Family Medicine Department of our institution to develop more services using the ECG and HRV data, and we plan to extend our research to other vital signs, such as respiration and SpO_2.

Acknowledgment

This work was funded by the Korean Ministry of the Knowledge Economy (#10033321).

References

[1] SIIA, Software as a service: strategic backgrounder, Software & Information Industry Association, 2001.
[2] S.H. Hung, C.S. Shin, J.P. Shieh, C.P. Lee, Y.H. Huang, Executing mobile application on the cloud: Framework and issues, Computers and Mathematics with Applications 63 (2012) 573–587.
[3] A.J. Jameela, J. Imad, A.D. Alyaziyah, A.A. Fatmah, Security middleware approaches and issues for ubiquitous applications, Computers and Mathematics with Applications 60 (2010) 187–197.
[4] R.A. Schrenker, Software engineering for future healthcare and clinical systems, Computer 39 (2006) 26–32.
[5] M. Bell, Introduction to service-oriented modeling, in: Service-Oriented Modeling: Service Analysis, Design, and Architecture, Wiley & Sons, 2008, p. 3.
[6] M. Bell, SOA Modeling Patterns for Service-Oriented Discovery and Analysis, Wiley & Sons, 2010, pp. 390.
[7] T. Erl, Introduction SOA, session 3.4 common tangible benefits of SOA, in: Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall PTR, 2008, (Chapter 3).
[8] J. Roy, A. Ramamujan, Understanding Web Service, IT Professional, 2001, pp. 69–73.
[9] CORBAmed Roadmap, Version 2.0: OMG Document CORBAmed/2000-05-01, 2000.
[10] OMG Healthcare DTF website [online]. Available: http://healthcare.omg.org/papers/Services_Standards_Collaboration_Concept-v1.01.pdf (15.10.11).
[11] C.P. Cheong, C. Chatwin, R. Young, An SOA-based diseases notification system, in: Proc. on ICICS, 2009.
[12] Chih-Jen Hsu, Telemedicine information monitoring system, in: HealthCom 10th International Conference, 2008, pp. 48–50.
[13] J. Mykkanen, A. Riekkinen, M. Sormunen, H. Karhunen, P. Laitinen, Designing web services in health information systems: from process to application level, International Journal of Medical Informatics 76 (2007) 89–95.
[14] D. Linthicum, Leveraging the heritage—approaches to integrating established information systems, Intelligent EAI, CMP Media LLC, 2003.
[15] Object Management Group: Object Constraint Language OMG Available Specification Version 2.0 Formal/06-05-01, OMG, 2006, p. 5.
[16] Object Management Group: Object Constraint Language OMG Available Specification Version 2.0 Formal/06-05-01, OMG, 2006, p. 159.
[17] J. Warmer, A. Kleppe, The Object Constraint Language: Precise Modeling with UML, Addison-Wesley Longman Publishing Co., Inc., 1998.
[18] L. Ol'khovich, D.V. Koznov, OCL-based automated validation method for UML specification, Programming and Computer Software 29 (6) (2002) 323–327.
[19] P. Ziemann, M. Gogolla, Validating OCL specification with the use tool an example based on the BART cast study 80 (2002) 157–169.
[20] M. Benattou, J.M. Bruel, N. Hameurlain, Generating test data from OCL specification, in: ECOOP Workshop on Integration and Transformation of UML Models, 2002.

[21] T.W. Kim, T.G. Kim, Automated code smell detection and refactoring using OCL, Korea Information Processing Society 15-D (6) (2006) 825–840.
[22] M. Fowler, Refactoring: Improving the Design Existing Code, Addison-Wesley, 1999.
[23] Y. Kataoka, M.D. Ernst, W.G. Griswold, D. Notkin, Automated support for program refactoring using invariants, in: Int. Conf. on Software Maintenance, 2001, pp. 736–743.
[24] Polar website [online]. Available: http://www.polar.fi/en/products/tranning_software (05.10.11).
[25] Honeywell website [online]. Available: http://www.hommed.com/Products/LifeStream_platform.asp (06.09.11).
[26] Biocom Technologies website [online]. Available: http://www.biocomtech.com (10.10.11).
[27] LAXTHA website [online]. Available: http://www.laxtha.com/ (11.10.11).
[28] W3C website [online]. Available: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (08.10.11).
[29] C.H. Lin, Frequency-domain features for ECG beat discrimination using grey relational analysis-based classifier, Computers and Mathematics with Applications 55 (2008) 680–690.
[30] M. Lubell, S. Marks, Health fitness monitor, US Patent 4566461, 1986.
[31] M. Hirai, G. Masuda, ECG description in MFER and HL7 version 3, APAMI&CJKMI-KOSMI Conference, 9–2, 2003, pp. 338–339.
[32] B.D. Brown, F. Badlilini, HL7 aECG implementation guide final, in: Regulated Clinical Research Information Management Technical Committee, 2005.