A SOA and Knowledge-based Telemonitoring Framework: *Design, Modeling, and Deployment*

http://dx.doi.org/10.3991/ijoe.v9i6.3312

W. Zhang, K. Thurow and R. Stoll
University of Rostock, Rostock, Germany

***Abstract*—Telemonitoring systems have proven to greatly reduce medical costs while improving the quality of medical care. Today, the main factors restricting the development and popularization of Telemonitoring systems include scalability and compatibility. The current sensor devices lack unified standards and deliver complicated data structures. This paper presents the design for an ontology-based context model, and related middleware, that provide a reusable and extensible application framework for Remote Healthcare and assistance at home. We define the semantic information to describe attributes of sensors, services and working procedures. Developers may rewrite the service definition to adapt to new requirements as needed.**

***Index Terms*—telemonitoring; SOA; knowledge; modeling; ontology**

I. INTRODUCTION

A Telemonitoring System can be defined as a technological means for sending remote physiological information and medical signals through a communication network to a monitoring center for analysis and diagnostics. Remote monitoring systems generally include three components: a monitoring center, Telemonitoring device, and a communication network connecting the two. This system integrates applications and devices like medical sensors, smart phones and data-store centers to provide health care services.

Telemonitoring systems currently face many challenges, primarily because the sensor technology, Telemonitoring's key technology, is still in the early stages of development. In the fields of architecture and service, the system faces the following two technical challenges:

- Dynamistic. The types of medical sensing devices are varied and lack unified standards. The present Telemonitoring system is mostly researched and developed for a specific sensing device. In other words, an unique connection code and working process is designed for each type of sensor, greatly reducing the system's scalability and presenting a significant obstacle for the popularization of Telemonitoring systems today.
- Heterogeneity. The heterogeneity between the Telemonitoring systems and the commonly used medical systems like HIS, LIS, causes data and service interaction difficulties which may easily create an "information island".

Example: Home HeartRate monitoring is a typical application scenario of Telemonitoring.

1) First, the patient needs to wear a medical sensor that can measure his heartbeat.
2) The heartbeat data collected by the sensor is first transmitted to a local data processing center Gateway (usually a smartphone) which will send the data to a remote medical center.
3) The medical center will store the acquired data in the database then transfer the medical data to a data-process-module which returns the processed results to the smart phone client.
4) Under certain conditions, an alarm is triggered as a reminder to the patient.

This example shows that, to solve the above problems, a Ubiquitous Telemonitoring System should have the following three abilities:

- The system should have the ability to discover, integrate and control new medical sensors.
- The sensor should connect with the local data processing gateway and transmit corresponding physical parameters according to medical service needs.
- In the medical service center, there should be open interactive data interface services. All types of medical services and data processing services should be published using open standards so that they can be discovered and employed by external systems.

II. METHODS

A. Context-Aware Middleware

The first and second problem can be solved by establishing a Context-aware Middleware [1]. In order to extricate remote medical applications from tedious sensor data acquisition and management, this paper proposes a Telemonitoring Context-Aware Middleware (TCAM). With TCAM, the upper-layer applications would not be concerned about acquiring the context data, but only about the business logic itself.

As shown in Table 1, applications based on TCAM are divided into three layers. TCAM separates the collection of context information and the development of aware applications. It provides a Subscriber API (i.e. Subscriber Application Programming Interface [2]) for the developer of the upper-layer application; TCAM is then responsible for completing the data collection of various kinds of sensors, including physical, virtual and logic sensors.

TABLE I.
TELEMONITORING CONTEXT-AWARE MIDDLEWARE

Context aware applications	
Telemonitoring Context-aware Middleware	Subscriber API
	Pub/Sub framework
	Provider API
	Data collection
All kinds of sensors	

All Context-aware Middleware that exists today, suffers from a common defect; they do not provide support to external sensor data. A major advantage of TCAM, as proposed in this paper, is that it provides support for external sensor data. The Mobile Terminal as the gateway, utilizes the Experia-ARCS-BL/ANT module of Sony to communicate with external sensor nodes (via Bluetooth, ANT protocol) to collect its data. At the same time, in order to overcome the dynamic properties and the heterogeneity of the external sensor data, this paper presents the design for a sensor context bnowledge-base, based on ontology [3], and develops a data definition language (DDL) to dynamically resolve sensor data according to the information in the knowledge-base.

B. SOC and Modeling

The second problem can be solved using a form of Service Oriented Computing (SOC) [4] and Workflow Modeling [5]. The SOC pattern hides the underlying implementation of the service; the workflow combines different services to control the whole process. For example, the above remote heartrate monitoring scenario can be designed as a SOC model; the alarm-process can be published in the monitoring center as a type of published Webservice.

The remote medical SOC model in this context means to wrapped devices , medical tasks as a software services. With the use of SOC to acquire data from the device as a service, the bottom communication protocol details are hidden from the users. When writing the workflow, the users can directly use the Medical Web Services provided by the knowledge-base, as described above, without having to deal with device protocol issues. Since there is no need to write code with PERL, C or JAVA, this method significantly simplifies development of Telemonitoring systems; it allows the integration of new services or component devices which improves the compatibility of the system with external systems.

III. SYSTEM STRUCTURE

A. System structure

The system architecture is shown in Fig.1, From a physical perspective, it is based on a C/S architecture which includes the sensor terminal, smart phone client terminal, and remote server terminal. From a functional perspective, the whole system can be divided in to three modules: Client Management Modules (CMD), Server Management Modules (SMD), and communication mechanisms.

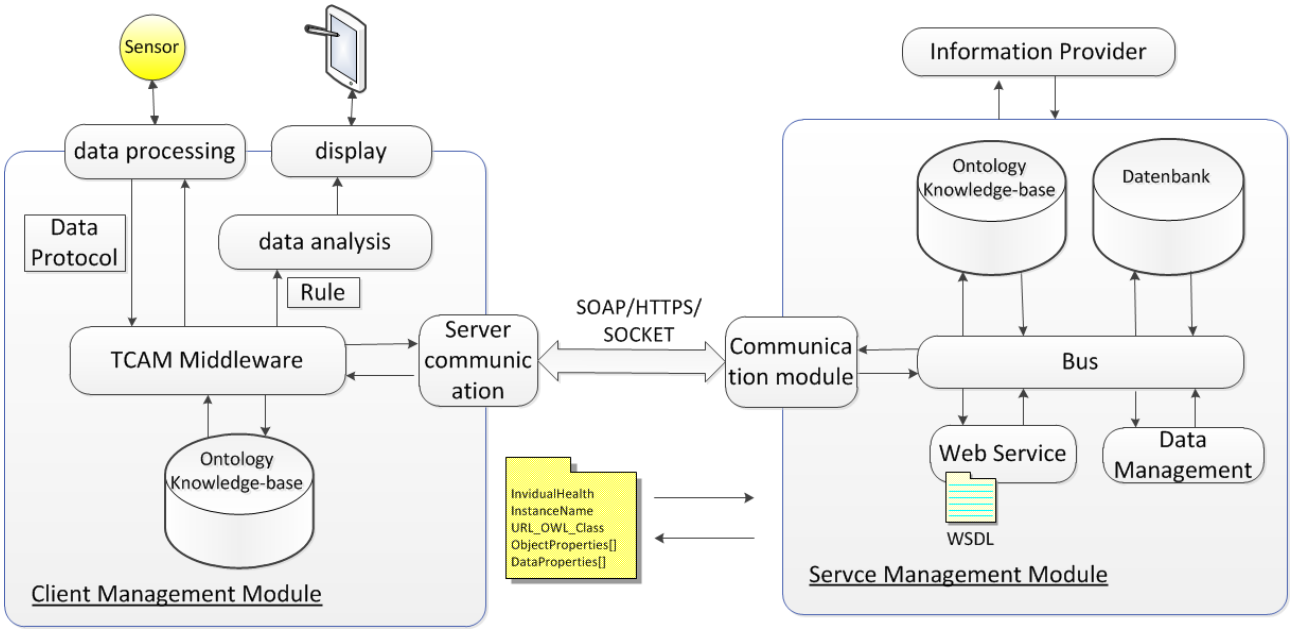

Figure 1. Telmonitoring System Architecture: Management Modules and Communication link

1. The core module at the smart phone client is the TCAM Middleware. It is responsible for communicating with the various sensors and performing data collection. Other core components include data processing, analysis, and display modules plus server communication modules. It should be noted that, unlike other Telemonitoring systems, the communication between client and server is bidirectional. In addition to sending data to a server, the client terminal can also synchronize information with the server. The design concept of the client terminal is to offload as much processing as possible to the server, so that the smart phone client functions only as gateway [6] to display and transfer data.

2. On the Server-side, the core component is the knowledge-base, based on ontology. Ontology means "formalized, clear and detailed specifications to a shared conceptual system." Ontology provides a kind of shared vocabulary, including object type or concepts, and their properties and relationships existing in special fields. In this project it is used to store Information and Ralations, including sensors, remote medical services, control strategies, and patients' personal domain knowledge. Other components on Server include a communication module that is responsible for transmitting information between the server and the client terminal, a data management module, plus a web service module. These modules communicate through a service bus.
3. As shown in Fig.1, the communication between CMD and SMD is implemented through HTTP/HTTPS, SOAP and SOCKET. HTTP is used in synchronous XML workflow description; SOAP is used to transmit ontology instance information; SOCKET connection is used in high real-time demand, for example, transmitting ECG (1024Byte/Sec) and Acceleration (200Byte/Sec) data.

B. TCAM Structure

The TCAM proposed in this paper is based on the Android system for design and implementation. Its architecture is shown in Fig.2,

According to the previously mentioned three-layer structure, TCAM include these three layers, i.e. context data acquisition, context data transmission and distribution, etc. TCAM first uses the context data collection module to collect the original context data provided by the sensors, then transmits the data to the upper sensing applications via the Pub/Sub framework. The Pub/Sub framework [7] completes the context data transmission and distribution.

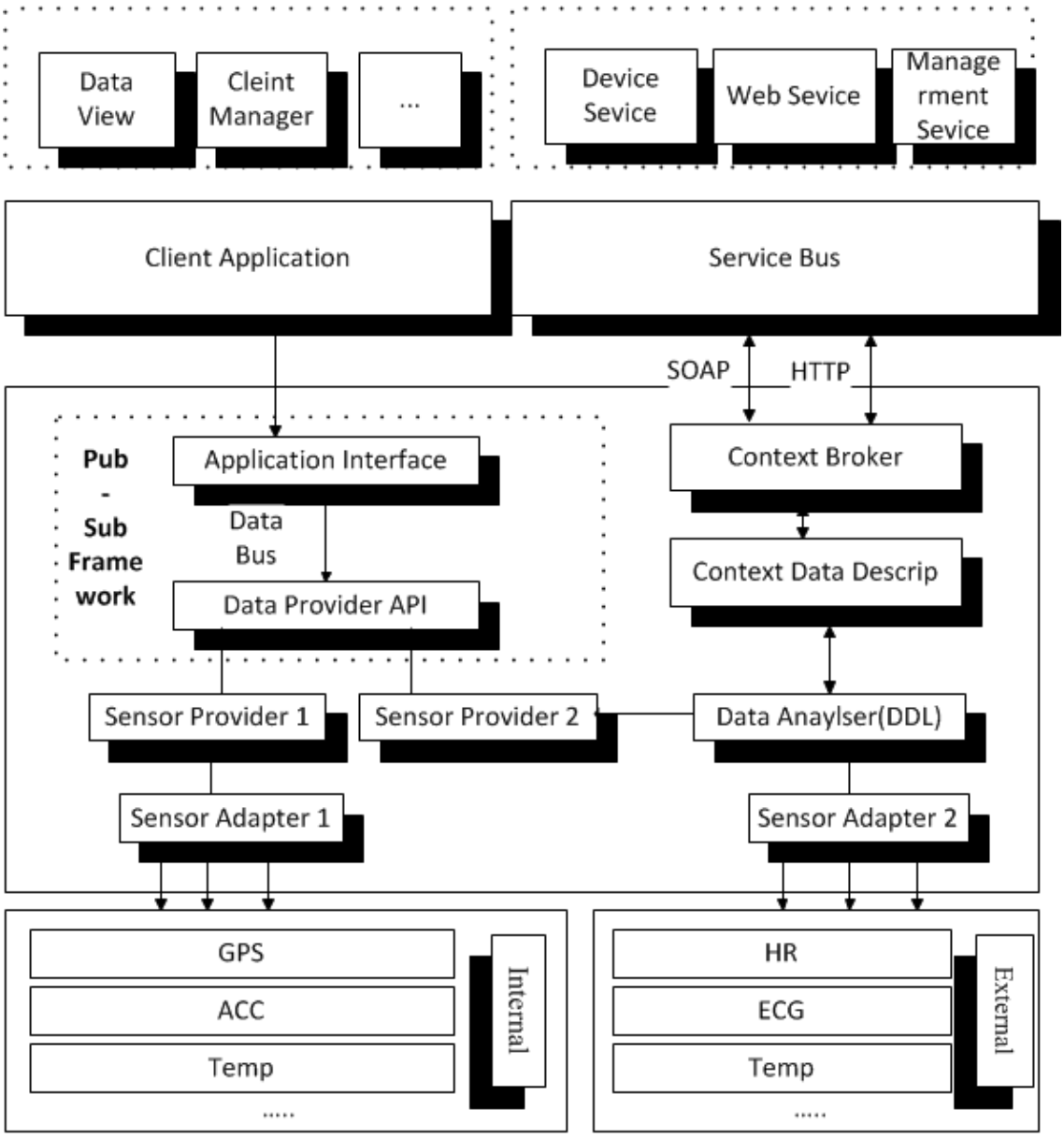

Figure 2. TCAM Architecture

Unlike common Middleware, this project adopts a distributed context knowledge-base and the server terminal conducts reasoning through the context instances obtained from the device and web services. The context information uses an inference engine based on conditions to conduct reasoning via OWL/RDF [8] modeling. Once the conditions are met, the context processing module will send information to the context-aware service in the client terminal through the SOAP protocol. Other important components of the TCAM include the data provider and the service bus.

1) Data Provider (Client)

Data provider(Internal) 1 and data provider(Externel) 2 are shown in Fig.2, A data provider usually exists in the form of a plug-in and is written using Provider API. After the providers are completed, the Provider API will transmit the data to the upper sensing applications. During the actual realization process, data provider 1 and data collector 1 are usually combined and exist as many different system plug-ins. However, data provider 2 and the data package resolver, are combined and exist as an independent plug-in. Under these circumstances, the data provider can directly transmit the data, read by the data collector, and the data analyzed, by the data package resolver, to the upper sensing applications via Data-Bus Reference [9] .

2) Service Bus (Server)

The service bus is designed to combine three kinds of services as follows:

- The service provided by the sensor device;
- The web service that can be directly visited via the network;
- The functional modules on the Server-Side, for example, the data-process-service, alarm service etc.

In addition to important functions like binding and routing of messages, the service bus can also provide service registration. Service registration allows other service modules to find and inquire web services and context processing services. All the context providers and context aware services need to be registered to the Bus through this registration module.

IV. SYSTEM DESIGN

A. System Operating Principle

In the last section, we introduced the system structure. To further specify how the system works, Fig.3, provides a flow chart that describes the establishment of remote health care service procedures, based on a knowledge base and workflow modeling. The outline of the design phase is as follows:

- Step-1:Ontology Knowledge Base Design. The medical experts and the developers should establish an Ontology Knowledge Base
- Step-2:Modeling. Establish a Telemonitoring workflow model based on a knowledge base, then define device and services in the workflow.
- Step-3:Code Generation. Convert the workflow model into executable code
- Step-4:Code Execution and Deployment.

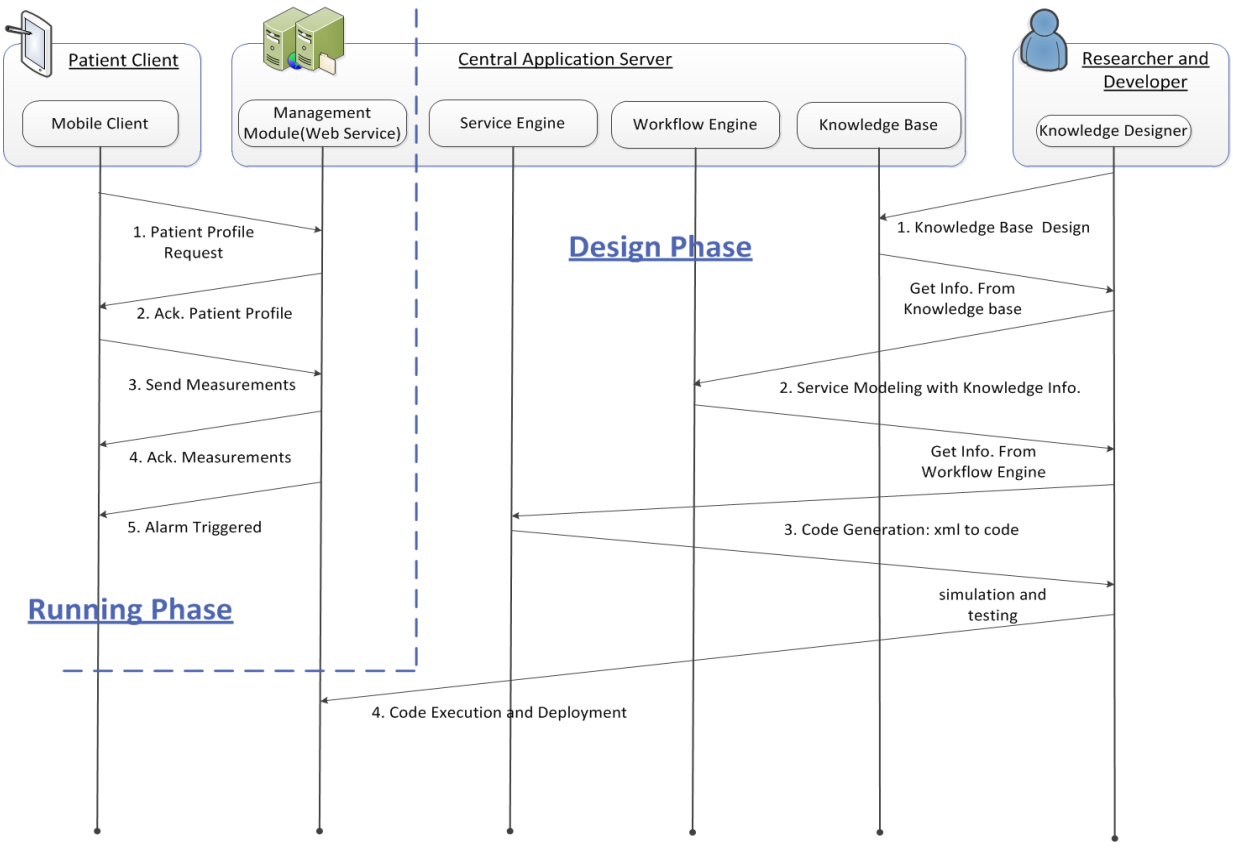

Figure 3. Remote Health Care Service Design and Running example

B. Knowledge-base design

The context knowledge-base design has two parts. The first part stores the ontology and their instances in the system. The other part provides a reasoning interface and context discovery, i.e., how to inquire and to add modules. Ontology instances may be pre-defined in the profile or may be retrieved from the device and other services. The pre-defined profile is dynamically imported into the system at the start of system operation. The context information [18] also includes some fixed information from a specific time; for example, the patient's blood type, emergency telephone number, clinical records, etc. At the same time, the context knowledge-base supports the inquiry and discovery of services/device based on semantic information [16].

The ontology used in this paper includes the following key parts:

- **Device ontology**: describes the concept of relevant device.
- **Function ontology**: describes the concepts related to service templates and functions.
- **Patient ontology**: describes the information related to the patient.

The corresponding ontology instances include:

- **Device instance**: records the device ID, type, function, status, position, energy consumption, brand and communication protocols.
- **Service instance**: is composed of a group of function ontologies.
- **Patient instance**: includes patient specific information and can function as the electronic health record (EHR).

This paper uses OWL to construct the model and process them using Sample Semantic Web Rule Language [19,20] . The reason for using the context to construct the model is that, it can not only express the context information, but can also use the inference engine based on conditions to conduct inference to the subtle context to get higher level information [10] .

The context models have the following forms: (Subject, Predicate, Value)

- **Subject** belongs to *S**: set of subject names. a patient, a place a smart phone, or a Emergency Service for example.
- **Predicate** belongs to *P**: the set of Predicate names, the patient's current body temperature or if the client terminal has connected to the service center, for example.
- **Value** belongs to *V**: the overall value of the Subject.

For example, we have a patient with ID 001, body temperature 37°C. Use a PDA to connect to the service center, and the information would be recorded as:

➢ *(Patient001, TemperatureFromPDA001, 37)*

Fig. 4, shows the ontology UML picture based on OWL.

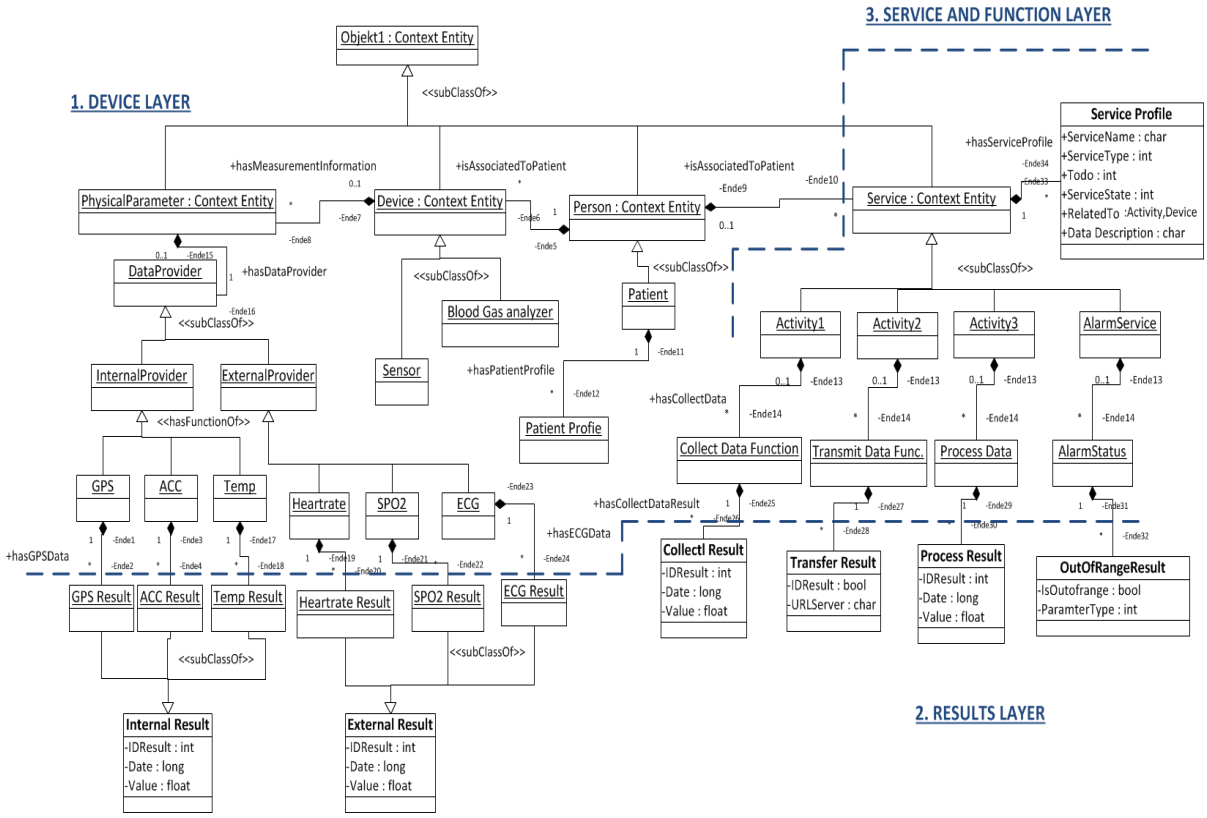

Figure 4. Telemonitoring Ontology UML Representation

Table 2 shows a few example rules, which the system uses to predict the sensor-data-specific service and Table 3 shows basic example policy we defined for each Emergency Service level [17]. The HeartRateHigh rule assigns the Emergency Service level when the heart rate is abnormally high. The hasRangeMin and hasRangeMin properties specify a normal heart rate in each instance of the HeartRateSensorData class. If a patient's heart rate is

greater than the specified normal maximum, then the system sets the heartrate alarm and start the Emergency Service [21].

In this paper, the service is also included in the ontology; the service will invoke specific activities. There are two advantages in doing this: the first advantage is that the specific details of the service and the service sub-actions are divided. Under these circumstances, the description of the service doesn't need to be changed when the device information, used by a sub-activity, changes. The other advantage is that the description of service by ontology, can make inference to the changes that should be made by the service; the inference processing depends on the service status, context information, and the custom rule. At the root node, is the design of a type of named service and a specific service level taking service. After inference is completed and the instances of service type exist, SOAP objects are sent to the service response components and then to the client terminal, including all the service information. We use the following attributes to describe a service:

a. ServiceName: the name of the service;
b. ServiceType: The type of service and function description information;
c. Todo: describes the current sub-activities of the service;
d. ServiceState: indicates the current state of a service. There are three values to be chosen: -CLOSE. , -INIT and -OPEN and they respectively indicate the service has not been started, or the service needs to be initialized or is started for use.
e. RelatedTo: indicates which specific function in the ontology the service is related to, and what kind of device instances the service is related to. If it is a common service, then the property is empty.
f. DataDescription: data description describes the data type and format that needs to be collected by the service. The information is stored in the service profile and uploaded when the system is operational.

TABLE II.
RULES FOR THE SYSTEM'S ALARM MANAGEMENT

RULE	*DESCRIPTION*
HeartRateHigh	(?patient rdf:type HeartRateSensorData) , (?par1 hasSensorDataResult ?v1) , (?par1 hasRangeMax ?Max) , greaterThan(?v1, ?Max) -> (?servicestate hasEmergencyService 'MES')
HeartRateLow	(?patient rdf:type HeartRateSensorData) , (?par1 hasSensorDataResult ?v1) , (?par1 hasRangeMin ?Min) , greaterThan(?v1, ?Min) -> (?servicestate hasEmergencyService 'MES')

TABLE III.
EMERGENCY SERVICE NOTIFICATION POLICY EXAMPLES

EMERGENCY SERVICE LEVEL	*NOTIFICATION POLICIES*
Low Emergency Service (LES)	- sensor alarm - sms to patient relative
Medium Emergency Service (MES)	- sensor alarm - sms , mail and call to relative
High Emergency Service (HES)	- sensor alarm - message to emergency operator - call to relative

C. Modeling

After the knowledge base design is complete, the next step is to construct a model for a specific remote medical service. The project uses three stages to design the composition of the service [11]:

1. Design of abstract workflow
2. Functional configuration
3. Device discovery

One specific problem needs to be solved in each stage: work procedures, function and device.

At the beginning, the developers design an abstract workflow, including steps and their designated relevance. Then, the developers connect the block in the workflow to the function of the service ontology. Finally, the function is mapped to the specific device detail via the knowledge- base. This paper uses PSML [22] as the modeling language as it allows modeling, composition, code generation, ontology, and analysis capabilities from a single language.

PSML is a modeling language that can be used for modeling, composition, analysis, simulation, and policy enforcement for service-oriented applications. The PSML ACDATER models, including Actor, Condition, Data, Action, Timing, Event and Relation, define the basic elements for applications. Among these model elements, Actors, Conditions, and Actions contain their internal workflows, and they can be used as a component in workflows. Workflows will be described by a set of behavior tags [11]. Fig. 5, shows a constructed abstract workflow that contains only a start point and two processes, after specifying the functions, the original empty block of the workflow can be used to describe/related to a service/device

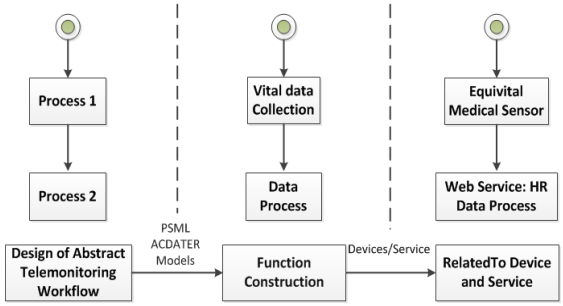

Figure 5. Three phases of service workflow construction

D. Code generation and service publication

After modeling, the service is executed in the operation environment, which requires converting the PSML model description to executable code. The method adopted in this paper is to first convert the PSML model into the more commonly used XML format, then convert the XML into the executable code. Commonly, the service described in the XML cannot be executed independently. It may be executed in the following two ways:

- **through the XML parser**
- **through the code generator.**

The first method, via parser, usually requires a complicated workflow engine that has domain knowledge. As such, the application in different fields requires different parsers and the implementation is comparatively complicated. The second method, with

code generation, requires an independently generated code for each application procedure. However, the code is lightweight. In the process of code generation, the main task is to design a code template then generate a new code template by changing code and codes for different target systems. Some code generation technologies using the current tools [11, 12] have been proposed. This paper uses the algorithm introduced in [11] to generate the code.

The first step is the translation; it serializes the PSML module into the XML document after completion of the device discovery step. The generating algorithm of the XML document is detailed as follows. It is the traversal algorithm of the BFS (breadth-first search) figure and uses one queue to record the nodes passed. When one node is pushed out of the queue, the corresponding content will be written into the XML document. At the same time, the subsequent unprocessed nodes will also be calculated. Under these circumstances, they will be pushed forward in the queue to conduct further processing. The output of the algorithm is an XML standardized workflow description document [11].

The second step is the code generation step. Each factor will first be pushed into the code generator and then the code generator generates the target code on the basis of the CodeSmith [15] template. The template is the executable document and it converts the XML description into executable code. Due to the platform independence of XML, the code generated during the code generation process can be adapted to different operating systems.

Algorithm : XMLGeneration algorithm

Input : XmlWriter xmlwriter, ExcutableElemenet start
Result : An XML file with all the workflow information

```
1  if start ! = null then
2  |  Queue queue = new Queue();
3  |  Queue.Add(start);
4  |  Set processedSteps = new Set(); next = start;
5  |  While queue.Count ¿ 0 do
6  |  |  next = (Step)queue.Dequeue();
7  |  |  if processedSteps.Contains(next.Id) then
8  |  |  |  continue;
9  |  |  end
10 |  |  Iterator iter = next.next();
11 |  |  While iter.has Next() do
12 |  |  |  queue.Enqueue(iter.next());
13 |  |  end
14 |  |  if next is StepAssign then
   |  |  GenerateStepAssign((StepAssign)next,xmlwriter);
15 |  |  else if next is StepExcute then
   |  |  GenerateStepExecuter((StepExecuter)next,xmlwriter);
16 |  |  else if next is StepExcute then
17 |  |  |  GenerateStepExecuter((StepExecuter)next,xmlwriter);
18 |  |  end
19 |  |  processedSteps.Add(next);
20 |  end
21 end
```

E. Deployment and execution

The deployment of the service is completed by experts at the remote terminal and the parsed code is transmitted to the client terminal of the smart phone via SOCKET connection. The mobile client will dynamically load the code and parameters to complete the initialization of the local service according to the service parameters. The service parameters include the collection of parameters, sensor connection and data from sensors according to communication protocols, etc. The client terminal of the smart phone exchanges information with the server and the message bus allows the interaction between server code and client terminal device.

V. EXPERIMENT

In this section we will use an emergency assistance service as a practical example to describe the above design pattern: how new devices are added to the ontology library and medical treatment begins via conditional reasoning.

The specific scenario for an emergency assistance service is as follows: An anomaly has been detected for a patient suffering from heart disease or high blood pressure; based on system reasoning, emergency assistance is requested. According to this service profile, the response time of the service is one minute, meaning that medical personnel must respond within one minute, react to the patient's condition and initiate corresponding measures. After the response, the service's mode is set to CLOSED. If the patient's condition remains abnormal, the service is started again.

A. The first step is to add new devices: In the experiment we used the wearable multi-parameter sensor Equital001 produced by HIDAGO.

When a new device is connected to the TCAM sensor adapter, context information must be converted into ontology instances before being sent to the knowledge base. The data gathering webservice inquires whether classes with identical names already exist in the ontology library based on the <deviceType> in the device description file. If not, a new class is added with the new <deviceType>. The following example shows the class of an Equital device being added:

- *<owl: Class rdf: ID = "Equital">*
  <rdfs: subClassOf rdf: resource = "#Device"/>
  <owl: disjointWith rdf: resource = "#"/>
  </owl: Class>

The service description file of middleware devices is searched to retrieve the stateVariable. This value is then compared to the current ontology library (knowledge base) to determine if there are any classes with this name. If not, a new class is added which acts as subclass of PatientData. Below the classes HeartRate and BloodPressure are added for the Equital device.

- *<owl: Class rdf: about = "#EquitalHeartRateValue">*
  <rdfs: sublassOf rdf: resource = "#PatientData"/>
  < owl: disjointWith rdf: resource = "#"/>
  < /owl: Class>
  <owl: Class rdf: about = "#EquitalBloodPressureValue">
  < rdfs: sublassOf rdf: resource = "#PatientData"/>

< owl: disjointWith rdf: resource = "#"/>
< /owl: Class>

With the establishment of the above-mentioned ontology, ensures that the device description file and the ontology name match. When the control point receives data from the device, it looks for the parameters in its own device files and device service files and creates an instance. The above-mentioned Equital device might, for example, receive event messages from the control point detailing HeartRate (Beats/Minute) = 60, BloodPressure (systolic, mmHg) = 120. In this case the practical example would look as follows:

- *<EquitalHeartRateValue rdf: about = "#EquitalHeartRate001">*
  <hasHeartRateValue rdf: datatype = "&xsd; double"> 60.0</hasHeartRateValue >
  < /EquitalHeartRateValue>
  < EquitalBloodPressureValue rdf: about = "#EquitalBloodPressure001">
  < hasPostureValue rdf: datatype = "&xsd; double">120.0</hasPostureValue >
  < /EquitalBloodPressureValue >
  <Equital rdf: about = "#Equital001">
  < provideData rdf: resource e= "#EquitalHeartRate001"/>
  <provideData rdf: resource = "#EquitalBloodPressure001" />
  < /Equital>

After the context information is converted, the data gathering webservice will send the aforementioned information to the knowledge-base.

B. The second step is to execute the service.

When the middleware is executed, the system automatically generates an instance for every service of a class, which is saved in the knowledge-base. The important information in these service instances is as follows:

- *<EmergencyService rdf: about="#service001">*
  <Num>100< /Num >
  <serviceState>CLOSE</serviceState>
  <todo rdf: resource= "#Alarming" />
  <todo rdf: resource ="#PrepareEmergencyOperator"/>
  </EmergencyService >

With the previous scenario in mind, we can define the specific conditions for the emergency service to be triggered: if the heartbeat of the patient drops below 40, or the blood pressure (systolic) exceeds 170. Reasoning may be carried out according to the rules of the language itself, for example:

- *(?a ?p ?b), (?p rdf: subPropertyOf ?q) -> (?a ?q ?b)*

User defined rules may also be added in the reasoning process, e.g.:

- (? patient, hasBloodPressureValue, vl), GE(?vl, 170), (? patient, hasHeartRateValue, vl), LE(?vl, 40) -> (? patient patientState "danger")
- (?patient, patientState, "danger"),(?Patient, hasDisease, "movementfail"), (EmergencyService, ServiceState, "CLOSE") -> (EmergencyService, ServiceState, "OPEN")

The reasoning process is as follows:

1) On system start, generate instances for all services and introduce a service profile.
2) examine newly acquired information, determine if lower level context is consistent with context; if there are inconsistencies, terminate reasoning.
3) store new context information in knowledge-base.
4) deduce high level information according to own context knowledge.
5) determine if newly deducted knowledge is consistent; if there are inconsistencies, terminate reasoning.
6) store newly deducted knowledge in knowledge-base.
7) go through newly deducted knowledge, determine service mode. If a service needs to be started, trigger start and send service information.
8) renew service mode instances in knowledge-base.
9) terminate.

C. Deployment and Code generation

The deployment configuration includes at least two Context Manager nodes as show in Fig.6: We used a SonyEricsson-Xperia smartphone as the Client Manager (CM) node, connect with medical sensor module Equivital; and Server Manager (SM) node, which is deployed on a Central Application Server, in the domain of the organization which is responsible for the integration and delivery of Emergency Web Service.

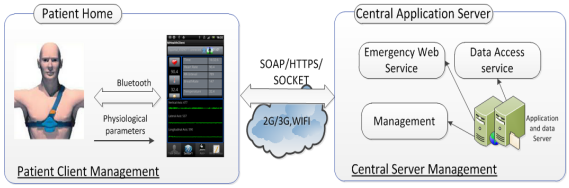

Figure 6. mHealth Prototype System Architecture

The Home Smartphone hosts the Client and communicates with the Central Application Server via 2G/3G/WiFi. The Central Application Server hosts the Server SM and other assistance and information service applications. These services include: a Emergency Web Service, patient record data management, and a web application which offers web-based access to health operators via mobile device.

Fig.7 describes the code generation process of the service. This paper uses the CodeSmith tool to generate code. CodeSmith first reads the XML document generated from the PSML model, then outputs the executable Java code. It can generate different code for different systems, according to the CodeSmith template. Finally, the code will be deployed in the server terminal for execution and synchronization with the client terminal. Fig.8 shows Online visualization of the incoming raw data.

Figure 7. Example code generation from a workflow model to XML representation to Java code

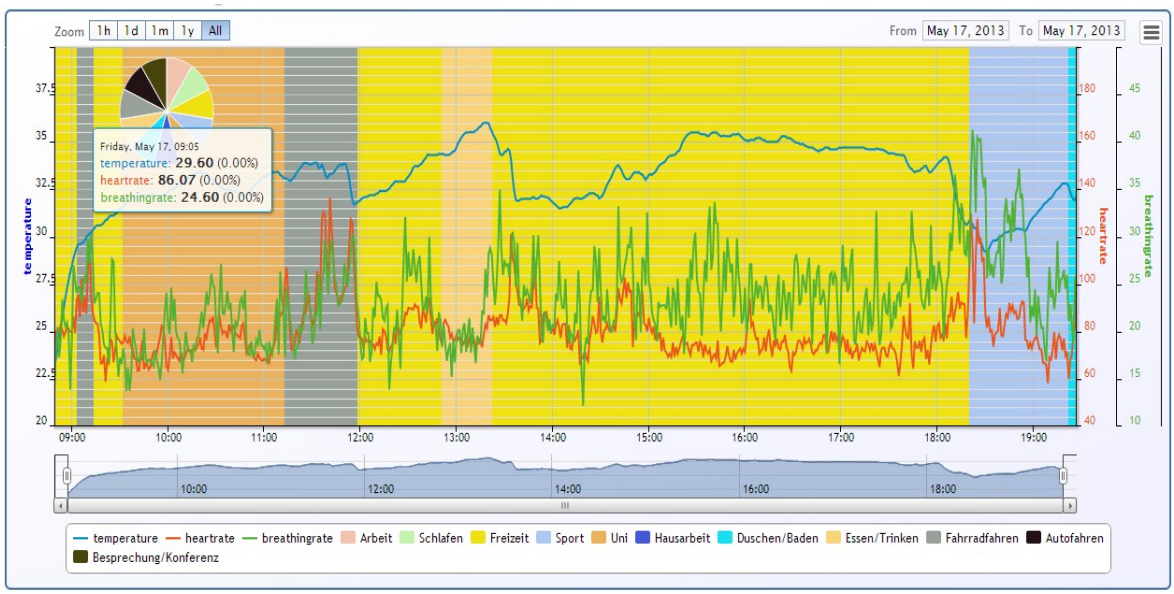

Figure 8. Online visualization of the incoming physiological data

VI. CONCLUSION

This paper designs an innovative Telemonitoring framework based on services and ontology technology. All steps in the Telemonitoring are modularized to facilitate dynamic trend, discovery, code generation and deployment of new remote medical services through semantic ontology information. The actual experiments in the operation indicate that the services-oriented design can greatly reduce the complexity of new application development. A context-aware Middleware is developed to support the scalability of the telemedical system; new sensor device and data formats can be integrated into the system conveniently. In the context reasoning module, ontologies are used to construct target services and circumstances. This paper, however, does not address the handling of context conflicts in the process of constructing ontologies and reasoning conditions. Further research will focus on the reliability and energy consumption problem of remote medical systems. Concerning the energy consumption problem, some new ultra low power transmission protocols, like BLE and ANT, have appeared that can greatly improve the sustainability of remote monitoring. Other challenging services will include the development of an intelligent error discovery and system recovery mechanism to maintain the stability of the system over time.

VII. ACKNOWLEDGMENT

This work is part of the eHealth-MV (Mecklenburg Vorpommern) project which results from the cooperation between University of Rostock Institute for Preventive Medicine and Center for Life Science Automation Germany. The authors thank the ministry of economy, work and tourism in Mecklenburg-Vorpommern Germany for the financial support of this project.

REFERENCES

[1] Baldauf M，Dustdar S，Rosenberg F. "A survey on context-aware systems," International Journal of AdHoc and Ubiquitous Computing, 2007, pp. 263-277 . http://dx.doi.org/10.1504/IJAHUC.2007.014070

[2] Papapanagiotou I, Falkner M, Devetsikiotis M. "Optimal functionality placement for multiplay service provider architectures," IEEE Transactions on Network and Service Management Volume 9, Issue 3, 2012, pp. 359-372 http://dx.doi.org/10.1109/TNSM.2012.061212.110032

[3] Elston J, Tsai WT, Li W, Bucur L. "Software architecturewith ontology for intelligent building management," In: Proceedings of international conference on control systems and computer science, vol 2, 2011, pp. 682–686

[4] Aziz M.W, Mohamad R, Jawawi D.N.A, Mamat R. "Service based meta-model for the development of distributed embedded real-time systems," Real-Time Systems, 2013, pp. 1-17

[5] Tick J. "Business Process based initial modeling at software development," SAMI 2013 - IEEE 11th International Symposium on Applied Machine Intelligence and Informatics, Proceedings. 2013, pp. 141-144

[6] Kuna M, Kolaric H, Bojic I, Kusek M, Jezic G. "Android/OSGi-based Machine-to-Machine context-aware system," Proceedings of the 11th International Conference on Telecommunications, ConTEL, 2011, pp. 95-102

[7] Wang N, Yang Z, Yang Y. "Based on event-driven and service-oriented architecture business activity monitoring design and implementation," International Conference on System Science, Engineering Design and Manufacturing Informatization, ICSEM. 2011, pp. 241-245

[8] Armas R, Cuenca G, Horrocks I. "MORe: Modular combination of OWL reasoners for ontology classification," Lecture Notes in Computer Science. 2012, pp. 1-16 http://dx.doi.org/10.1007/978-3-642-35176-1_1

[9] XIONG LB，NIU J，ZHANG JF，CHEN CF，SHEN XP. "Context Sensing Middleware Based on Mobile Devices. Journal of Chinese Computer Systems," 2011, pp. 1170-1174

[10] Cheng J, Rao R. "A context-aware middleware for pervasive healthcare," Computer Applications and Software, 2009, pp. 50-53

[11] Wu L, Yann-hang L, Wei-tek T, Jingjing X, Young-sung S, Jun-hee P, Kyung-duk M. "Service-oriented smart home applications: composition, code generation, deployment, and execution," Service Oriented Computing and Applications. 2012, pp. 65-79

[12] Cai Y, Zhao L. "Study on custom service combination based on BPEL," Advanced Materials Research. 2013, pp. 2451-2456

[13] Lee YH, Li W, Tsai WT, Son YS, Moon KD. "A code generation and execution environment for service-oriented smart home solutions," In: International conference on service-oriented computing and applications 2009

[14] Tsai WT, Fan C, Chen Y, Paul R. "Ddsos: a dynamic distributed service-oriented simulation framework," In: Proceedings of the 39th annual symposium on simulation, IEEE Computer Society. 2006, pp.160–167

[15] Ma TM, Zheng BL, Chai TY. "Oriented-service simulation platform for iron & steel production planning," Xitong Fangzhen Xuebao / Journal of System Simulation, April 2010, pp. 890-894

[16] Lasierra N, Alesanco A, García J. "Home-based telemonitoring architecture to manage health information based on ontology solutions," Proceedings of the IEEE/EMBS Region 8 International Conference on Information Technology Applications in Biomedicine, ITAB. 2010

[17] Lasierra N, Alesanco A, García J. "An ontology approach to manage individual patient profiles in home-based telemonitoring scenarios," Proceedings of the IEEE/EMBS Region 8 International Conference on Information Technology Applications in Biomedicine, ITAB, 2010

[18] Paganelli F, Giuli D. "An ontology-based context model for home health monitoring and alerting in chronic patient care networks," Proceedings - 21st International Conference on Advanced Information Networking and Applications Workshops/Symposia, 2007, pp. 838-845

[19] Subramanian M, et al. "Novel Sensor Technology Integration for Outcome-Based Risk Analysis in Diabetes. Proc," First Int'l Conf. Health Informatics, vol. 2, IEEE CS Press, 2008, pp. 119–126.

[20] Horrocks I, et al., "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," World Wide Web Consortium (W3C), www.w3.org/Submission/SWRL, May 2004

[21] Benlamri R, Docksteader L. MORF: "A mobile health-monitoring platform", May 2010, pp. 18-25

[22] Tsai WT, Paul RA, Xiao B, Cao Z, Chen Y. "PSML-S: a process specification and modeling language for service oriented computing," In: The 9th IASTED international conference on software engineering and applications (SEA), 2005, pp.160–167

AUTHORS

Weiping Zhang received the Diploma degree in Computer Science from Technical University of Dresden, Dresden, Germany, in 2009.

Since 2011, he has been a Research Scientist with the Institute of Preventive Medicine, University of Rostock, Rostock, Germany. His research interests include process information management systems and real-time mobile measurements of physiological parameters.

Kerstin Thurow studied chemistry with the University of Rostock, Rostock, Germany, and received the Graduate degree. Afterward, she received the Ph.D. degree from the Ludwigs-Maximilians-University, Munich, Germany (under the guidance of Prof. Lorenz), working on metal–organic sulphur compounds. In 1999, she received the Habilitation degree from the Department of Electrical Engineering, University of Rostock.

Dr. Thurow received the highly renowned Joachim-Jungius-Award of Science in 2004, in addition to many awards, such as for the foundation of a start-up company Amplius—Screening Technologies & Analytical Measurement.

Regina Stoll received the Dip.-Med. degree in medicine, the Dr.med. degree in occupational medicine, and the Dr.med.habil. degree in occupational and sports medicine from the University of Rostock, Rostock, Germany, in 1980, 1984, and 2002, respectively.

She is currently the Head of the Institute of Preventive Medicine, University of Rostock. She is a Faculty Member with the Medicine Faculty and Faculty Associate with the College of Computer Science and Electrical Engineering, University of Rostock. She also holds the Adjunct Faculty Member position with the Department of Industrial Engineering, North Carolina State University, Raleigh. Her research interests include occupational physiology, preventive medicine, and cardiopulmonary diagnostics.

Submitted 03 October 2013. Published as re-submitted by the authors 05 November 2013.