Remote Configuration and Deployment of Sensor Drivers for a Medical Bluetooth Sensor Gateway

Daniel Kümper, Ralf Tönjes
Faculty of Engineering and Computer Science
University of Applied Sciences Osnabrück
Osnabrück, Germany
{d.kuemper, r.toenjes}@hs-osnabrueck.de

Abstract— **Outpatient nursing services expend a vast amount of time for manual nursing documentation and management of patient basic claims data. Context-driven acquisition of sensor data, documentation of observations and their automated processing can reduce this time, while availability of documented information increases. Usage of wireless sensors is a major element of long-term measurement of medical patient context. Currently a technical experienced nurse, patient or an additional technical service provider is needed on site to configure new sensors and pair them with the patient's or nurse's monitoring device. To ease configuration and maintenance this paper proposes a technique for remote configuration of a sensor gateway and secure automated pairing with Bluetooth sensors. For security reasons this paper presents a generic sensor driver interpreting XML-based protocol specifications to execute the protocol thus avoiding the installation of additional sensor drivers in the operating system for each new sensor. The concept is proven by a theoretical analysis and by an evaluation of the implementation.**

Keywords- Bluetooth; PAN; remote configuration; auto configuration; driver deployment; sensor deployment; sensor delivery

I. INTRODUCTION

To support outpatient nursing services, the ContextCare project [1] is researching ways to use wireless health sensors for supporting documentation and alert services in the context of medical data. Currently nurses need a vast amount of time for manual nursing documentation and management of patient basic claims data. Hence the time available for primary care is decreased. The usage of networked medical sensors is a possibility to ensure accurate documentation that requires less time spent by the nurse [2] [3]. Wireless sensors can be connected to a patient-owned device that is able to inquire medical data and also communicates with the nurse's technical equipment. Available systems mostly support only one isolated application like diabetes, cardiology or patient alarming. Furthermore the isolated commercial solutions do not offer role based data access to multiple users at the same time although this is an essential qualification for public healthcare systems [4] [5]. Although wireless sensors offer a high flexibility in their usage, patients as well as nurses may have problems, setting up the sensor network. Compared to wire-based sensors wireless devices usually need additional configuration by interacting users. Additional security credentials (like encryption) are necessary because wireless sensors use a broadcasting medium [6]. In case of using Bluetooth it means scanning for new devices, selecting and securing the connection with a PIN code. For the usage of new devices interacting computers need a driver in the form of binary code or interpreter code that has to be deployed before a new sensor device can be used [7]. These drivers can either be distributed with the running software or can be reloaded or deployed by an update mechanism over the Internet if new sensors shall be connected to the sensor net. Because connecting to wireless sensors usually needs a very hardware- respectively system-near programming access, the software/system has to severely trust the distributed driver. In general, implementing a non-technical-user friendly sensor solution has to match two major requirements:

- New medical sensors must not need on site configuration by the patient or a nurse.
- New binary driver deployments for new types of sensors have to be avoided for security reasons.

Although a Bluetooth Health Device Profile (HDP) exists [8], most currently available sensors delivering medical data use the Serial Port Profile (SPP) with RFCOMM that emulates a serial cable to provide a simple substitution for existing RS-232 interfaces. Sparsely implementation of the HDP seems to result due to adaptation of older sensor modules containing simple serial interfaces and the more complex implementation of the HDP towards the usage of a simple self made protocol. The Bluetooth Service Discovery Protocol (SDP) allows devices to discover services supported by Bluetooth devices and examines their supported protocol stack. The SPP and SDP are supported by most operating systems for mobile devices, so driver specific differences for sensors only depend on their high-level communication protocol (Fig. 1).

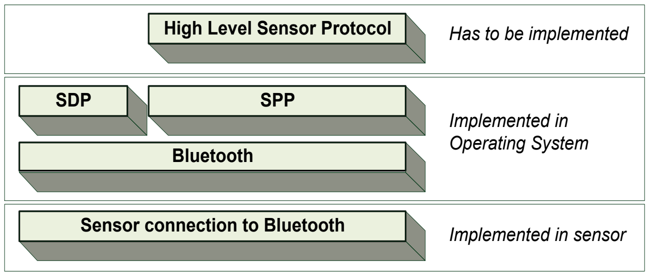

High Level Sensor Protocol
Has to be implemented
SDP
SPP
Bluetooth
Implemented in Operating System
Sensor connection to Bluetooth
Implemented in sensor

Figure 1. Communication Stack between Sensor and Receiver

II. Architecture

In order to support the healthcare documentation and accounting of medical services various Bluetooth sensors use a sensor gateway to connect to a Healthcare Management Server (HMS) which acts as a management and caching proxy for numerous sensor gateways and is accessible over the internet. In small scale scenarios and scenarios where a continuous wireless internet connection is ensured the HMS interface can directly be implemented on the sensor gateway. Technically the sensor gateway is a small device in the size of a mobile phone, capable of using Wireless-LAN and UMTS. It recognizes new measurements of paired medical sensors and abstracts sensor protocols to an XML-based document language describing the whole observation [9]. The HMS receives medical information as well as sensor gateway- and sensor-metadata. Serial numbers, time and the location of the measurement are documented to present the actual user context. Client software offers role based access to medical patient data. An Observation Library stores the medical chronicle of a patient in a database and processes XML-observations for more efficient value handling. A brief overview of the ContextCare architecture is presented in Fig. 2. In order to provide a platform independent open interface the ContextCare architecture communicates via REST (Representational State Transfer) based on HTTPS (Hyper Text Transfer Protocol Secure). Data is represented and transmitted in the form of XML (Extensible Markup Language)-documents. Authentication and authorization is ensured by PGP (Pretty Good Privacy)-based message signatures.

III. Remote Configuration and Secure Pairing

Medical devices for patients are commonly delivered by a medical device supplier that gets a prescription from a doctor and manages payment modalities with the health insurance. Using a centralized architecture for sensor management the supplier can also register a sensor to the patient by providing little information.

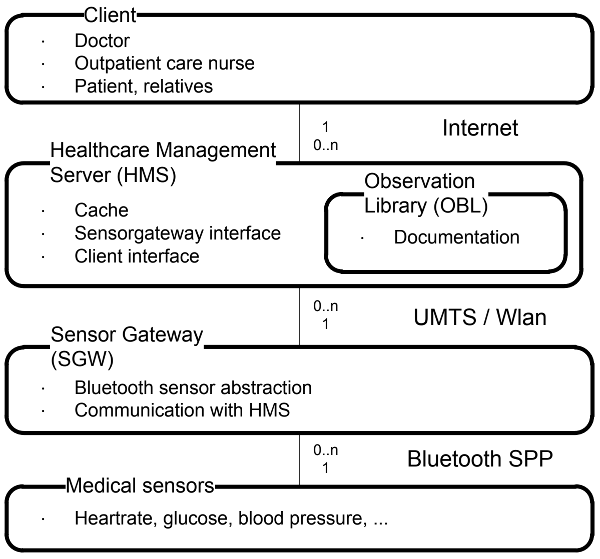


Figure 2. Architecture Overview

Important data for remote sensor registration are:

- Patient Identification
- Device Type
- Device Serial Number
- Device Security Credentials

By providing this information, the HMS can inform a sensor gateway that in the near future a new Bluetooth sensor will appear in its Bluetooth neighborhood and is supposed to deliver data if it will be successfully connected. The sensor gateway receives the data and is able to pair [10] with the Bluetooth sensor device when it notices its first appearance without need for user interaction on site. Measurements can now be initiated either by the Bluetooth sensor or the sensor gateway. Sensor gateway configuration documents use the following information (per device):

- Name of the Bluetooth device
- Unique number for device identification
- Security credentials to initiate the pairing-process
- Device type that specifies the driver that has to be used

After receiving a new configuration the sensor gateway can automatically connect to a new device if it is close to and visible by the Bluetooth module. On safeguarded operating systems like an unmodified Android it is not possible to override the manual input of a Bluetooth PIN through an application without modifying the devices firmware. However it allows the automated discovery and driver selection process, while the user just has to type in the PIN the software is simultaneously showing up on the display. Fig. 3 shows an exemplary communication if a new sensor is delivered and registered at the HMS.

IV. Remote Configuration of a Generic Driver

Binary driver deployment for new sensors forces the sensor gateway to download software potentially containing security risks after the initial installation of its software. Drivers usually need capabilities to access hardware near system layers in order to establish Bluetooth communication.

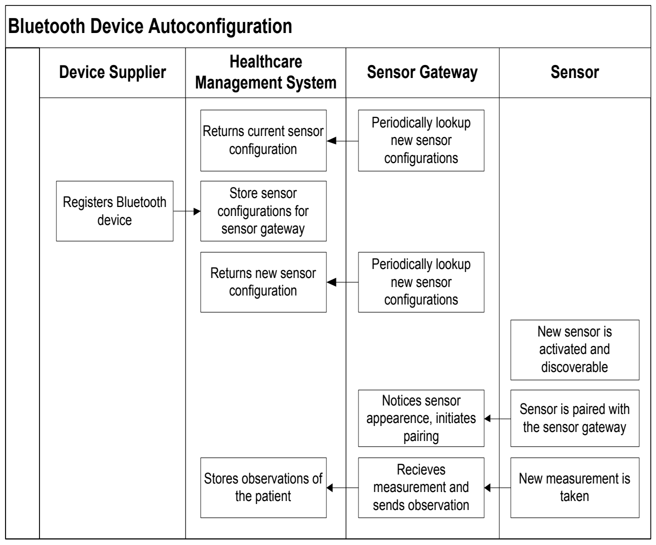

Bluetooth Device Autoconfiguration
Device Supplier
Healthcare Management System
Sensor Gateway
Sensor
Returns current sensor configuration
Periodically lookup new sensor configurations
Registers Bluetooth device
Store sensor configurations for sensor gateway
Returns new sensor configuration
Periodically lookup new sensor configurations
New sensor is activated and discoverable
Notices sensor appearence, initiates pairing
Sensor is paired with the sensor gateway
Stores observations of the patient
Recieves measurement and sends observation
New measurement is taken

Figure 3. Communication Diagram

A generic driver for Bluetooth SPP devices, which is configured by protocol description documents, minimizes the risks to potential bugs in a single driver used for hardware access. Furthermore it provides platform independent sensor drivers if generic drivers are implemented for target platforms. The following levels of complexity have to be supported for communication over one serial channel:

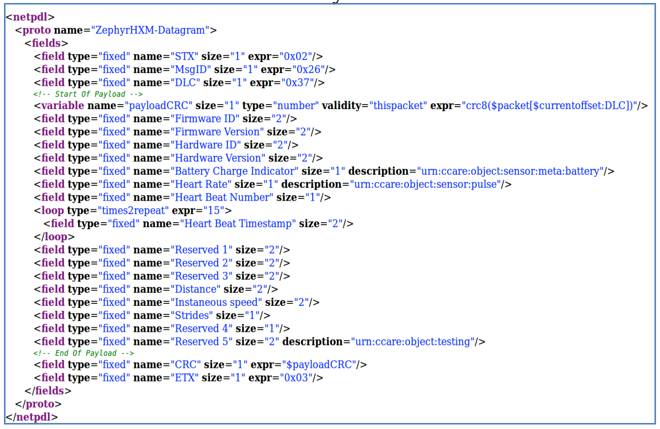
- The sensor is sending information packets without interaction needed. Datagrams have to be analyzed.
- The protocol requires control-related interaction like static start/ack commands.
- The protocol requires dynamic interaction where the "master" has to do some calculation for the data he sends to the sensor. For example a checksum over a command is computed or a data lookup like a PIN that has to be fetched from a database.
- The protocol is based on a state machine where commands depend on the actual state of the sensor.

XML documents enable a structural description of serial protocols. In contrast to proprietary plaintext protocols XML offers document validation using XSD (XML Schema). Furthermore XSD formally describes the available language elements. The solution presented in this paper divides the protocol description into two parts.

On the one hand datagrams have to be described to understand messages sent by the sensor or gateway. On the other hand a behavioral description of occurring protocol states and their reaction to new received datagrams has to be supported.

A. *Description of Datagrams*

Network Protocol Description Language (NetPDL) [11] is an application-independent packet format description language for effective description of complex packet header format and protocol encapsulation. It includes elements to describe fields, data types, and regular expressions. It also supports conditional elements and loops for dynamic length fields. To integrate it in the ContextCare architecture this paper proposes the description attribute of *field*-elements to describe the mapping of a field to its representing value (Fig. 4). The HMS uses Uniform Resource Name (URN)-tagged description fields to document the medical values as well as metadata like the current battery status of the sensor.



```xml
<netpdl>
  <proto name="ZephyrHXM-Datagram">
    <fields>
      <field type="fixed" name="STX" size="1" expr="0x02"/>
      <field type="fixed" name="MsgID" size="1" expr="0x26"/>
      <field type="fixed" name="DLC" size="1" expr="0x37"/>
      <!-- Start Of Payload -->
      <variable name="payloadCRC" size="1" type="number" validity="thispacket" expr="crc8($packet[$currentoffset:DLC])"/>
      <field type="fixed" name="Firmware ID" size="2"/>
      <field type="fixed" name="Firmware Version" size="2"/>
      <field type="fixed" name="Hardware ID" size="2"/>
      <field type="fixed" name="Hardware Version" size="2"/>
      <field type="fixed" name="Battery Charge Indicator" size="1" description="urn:ccare:object:sensor:meta:battery"/>
      <field type="fixed" name="Heart Rate" size="1" description="urn:ccare:object:sensor:pulse"/>
      <field type="fixed" name="Heart Beat Number" size="1"/>
      <loop type="times2repeat" expr="15">
        <field type="fixed" name="Heart Beat Timestamp" size="2"/>
      </loop>
      <field type="fixed" name="Reserved 1" size="2"/>
      <field type="fixed" name="Reserved 2" size="2"/>
      <field type="fixed" name="Reserved 3" size="2"/>
      <field type="fixed" name="Distance" size="2"/>
      <field type="fixed" name="Instaneous speed" size="2"/>
      <field type="fixed" name="Strides" size="1"/>
      <field type="fixed" name="Reserved 4" size="1"/>
      <field type="fixed" name="Reserved 5" size="2" description="urn:ccare:object:testing"/>
      <!-- End Of Payload -->
      <field type="fixed" name="CRC" size="1" expr="$payloadCRC"/>
      <field type="fixed" name="ETX" size="1" expr="0x03"/>
    </fields>
  </proto>
</netpdl>
```

Figure 4. NetPDL Description Heartbeat Sensor

When the sensor gateway receives a data packet from a sensor it is able to validate it with the NetPDL packet description and read out the values that are tagged with an URN. Packets sent from the sensor gateway to the sensor can also be built out of these documents. Therefore dynamic values (like a PIN) that need to be sent from the sensor gateway are looked up in the device or are requested from the HMS. Predefined fields with an expression can be helpful to identify the start and end as well as the validity of a packet. On a serial line no special layers or events exist which notice that a new packet is starting or a packet has ended. NetPDL provides declaration of variables inside datagrams. Furthermore the usage of common functions can be described if they are implemented in the sensor gateway. This feature is needed for protocol fundamentals like a checksum in a datagram.

B. *State Description*

NetPDL does not support the description of a protocol's temporal behavior. Therefore a mechanism like a protocol state machine is needed [12]. This paper employs State Chart XML (SCXML) [13] to describe the temporal behavior of the protocol. SCXML is a general-purpose, event-based state machine language that supports a clear behavioral description of reactive behavior. A serial protocol that we need to describe can be defined as a deterministic state machine. It is necessary to parse the datagrams for extraction of medical data as an output. Therefore it is required to specify a Mealy machine that defines an output function depending on the current state and the event/condition that leads to this state. The condition represents the datagram of which data is extracted.

The following formal definition of a Mealy machine with the 6-tuple ($\Sigma, \Gamma, S, s_c, \delta, \omega$) shows that all properties needed can be described by a subset of the SCXML language where:

- Σ is the input alphabet, a finite, non empty set of events that can be received
  - for example: Datagram received/sent, timeout, datagram validated/invalid
- Γ is a finite, non empty set of output values
- S is a finite, non empty set of communication states
- s_c is the initial state, an element of S (connection established, no bytes sent yet)
- $\delta : S \times \Sigma \rightarrow S$ is the transition function depending on datagrams and their values
- $\omega: S \times \Sigma \rightarrow \Gamma$ is the output function that depends on a state and the input event that has been received (gateway output depends on sensor datagram)

Table 1 shows that each property can be described by SCXML-tags combined with NetPDL-validation as conditions for events.

SCXML provides the ability to define a data model as part of the SCXML document. A data model consists of a *<datamodel>* element containing one or more *<data>* elements, each of which may contain a NetPDL datagram description. With a delay attribute it is possible to configure timeouts or to wait for trigger delays that initiate an event.

TABLE I. STATEMACHINE TO SCXML TRANSFORMATION

	SCXML transformation	SCXML example
Σ	*event* or *condition* attribute of a *transition* element	*event="event.received"* *cond="isValid(hxmPacket)"*
Γ	*expr*, the content of a *send* element	*expr="$hxmDatagram"*
S	element describing the *state*	*<state id="connected"> ... </state>*
s_c	*initial* element, child of root element refers directly to a state	*<initial><transition target="connected" /> </initial>*
δ	*transition*, a child of the *state* element can have *event* and *cond* attributes, describes next element	*<transition target="validated" />*
ω	*send*, a child of the state element can have *event* or *expression* attribute and defines the output	*<send>target="hms" expr="hms"</send>*

The SCXML Document describes not the protocol itself but a sensor gateway centric view for conducting communication with the sensor. Valid targets for the output function are:

- *sensor*: describes the communication outgoing to the Sensor
- *hms*: describes sensor data that will be sent to the Healthcare Management System

Fig. 5 shows the simple SCXML based protocol description of a Zephyr HXM Bluetooth heart rate sensor, which periodically pushes datagrams to the sensor gateway after the Bluetooth SPP connection, is established. It needs no bidirectional communication. The underlying state machine that is used to handle the communication protocol is described in Fig. 6.

```
<scxml version="1.0">
  <datamodel>
    <data name="hxmDatagram">
      <netpdl>
        <proto name="ZephyrHXM-Datagram">
          <!-- Datagram Description in NetPDL (see Figure 4)-->
        </proto>
      </netpdl>
    </data>
  </datamodel>
  <initial>
    <transition target="connected"/>
  </initial>
  <state id="connected">
    <!-- one packet should be received every second -->
    <transition event="receiving" target="received"/>
    <transition target="disconnected" delay="5s"/>
  </state>
  <state id="packet received">
    <onentry>
      <!-- building local data model for state -->
      <assign location="$hxmDatagram" expr="$_event.data"/>
    </onentry>
    <!-- first matching condition in document is taken -->
    <transition cond="validateCurrentPacket($hxmDatagram, ZephyrHXM-Datagram)"
    target="packet interpreted"/>
    <transition target="packet discarded"/>
  </state>
  <state id="packet interpreted">
    <onentry>
      <log expr="send current Observation"/>
      <send>
        event="sendUrnData" target="hms" expr="$hxmDatagram"
      </send>
    </onentry>
    <transition target="connected"/>
  </state>
  <state id="packet discarded">
    <transition target="connected"/>
  </state>
  <final id="disconnected"/>
</scxml>
```

Figure 5. Example of a SCXML Based Protocol Description

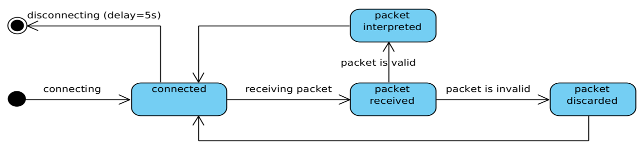


Figure 6. State Chart for Simple Data Protocol

V. EVALUATION

The sensor gateway prototype currently offers a limited set of events and functions implemented in its software. Auto configuration and driver deployment work for common medical sensors measuring glucose, blood pressure and heart rate, which the project currently uses. At the moment there are no sensors used that need calculation of fields in the datagrams which the sensor gateway is sending to a sensor.

The implementation of the generic Bluetooth driver for an Android sensor gateway is used to show its practicability. Android is a free operating system for mobile devices. The sensor gateway software can build the specialized driver, based on the XML description of the communication at runtime. Drivers for already configured and paired devices are built at program startup. For devices that are added to the sensor gateway while it is in operation, the XML document can be loaded from the HMS and a driver can be built during runtime. Fig. 7 shows this process.

XML-based drivers have got to be similarly efficient processing datagrams to be a suitable replacement for a device-specific binary driver. A high computing effort for generating the drivers during startup or while adding new sensors can be accepted.

To show the performance of the packet processing the datagram description of a ZephyrHXM device is used. ZephyrHXM is a Bluetooth based heart rate sensor, using SPP for communication. It delivers datagrams with a frequency of 1HZ. Datagrams consists of 60 bytes; describing 35 data fields (shown in Fig. 4). In order to evaluate the scalability, fictional XML-descriptions and datagrams of 120, 240, 480, 960 and 1920 bytes have been used in the test cases to determine the algorithm's time complexity. Measurement is performed in an emulated environment. Datagrams are provided by a test component, so radio effects and the sensor submission speed do not affect the measurement.

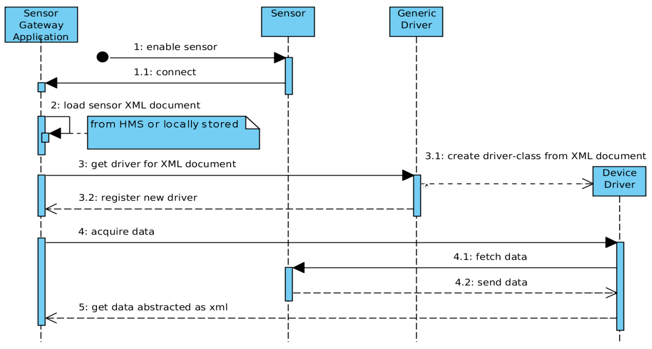

Sensor Gateway Application
Sensor
Generic Driver
1: enable sensor
1.1: connect
2: load sensor XML document
from HMS or locally stored
3.1: create driver-class from XML document
3: get driver for XML document
Device Driver
3.2: register new driver
4: acquire data
4.1: fetch data
4.2: send data
5: get data abstracted as xml

Figure 7. Generation and Usage of the Generic Driver

The histogram in Fig. 8(a) shows the generation time distribution of 100 device drivers (ZephyrHXM with 35 Fields, 60 bytes), generated out of the XML document. The mean generation time is 23.55ms. Fig. 8(b) shows the measured mean values of 100 driver generations each for the fictional, larger datagrams. This illustrates that the generation-algorithm used in the generic XML-driver has a linear time complexity O(n).

To determine the sensor data processing speed of the generated drivers, 100000 datagrams of each size are parsed by the corresponding generated driver. Fig. 9(a) shows the histogram of processing time for the generated ZephyrHXM driver. Due to scattered time peaks, caused by system components like the garbage collector, there are infrequent measurements (<0.5%) that take the multiple amount of time for processing (2000µs up to 100ms). These are cut off in Fig. 9(a) for better perception of the processing time distribution but are visible in Fig. 10.

The interruptions of the datagram processing thread caused by system components of the Android emulator are reproducible and determined with the Android profiling tool *traceview*. A plot of 10000 measurements, each represented by a blue dot, is shown in Fig. 10. Scattered time peaks clearly accentuate from the majority of realized measurements and represent clusters visible in Fig. 9(a).

Fig. 9(b) shows the processing time median for the datagram parsers. This illustrates that the parsing-algorithm also has a linear time complexity O(n) which is important to enable communication with more complex medical sensors like ECGs.

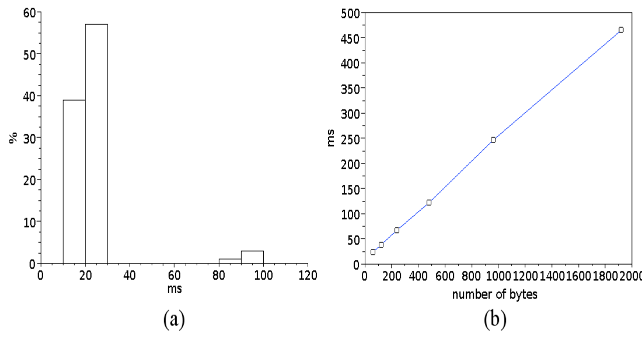


Figure 8. Driver Generation/Build Time

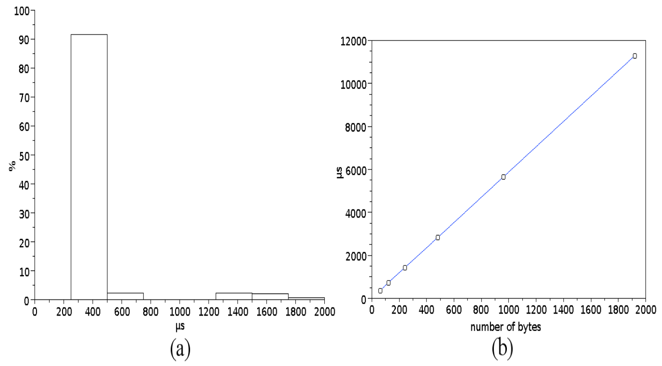


Figure 9. Datagram Parsing Time

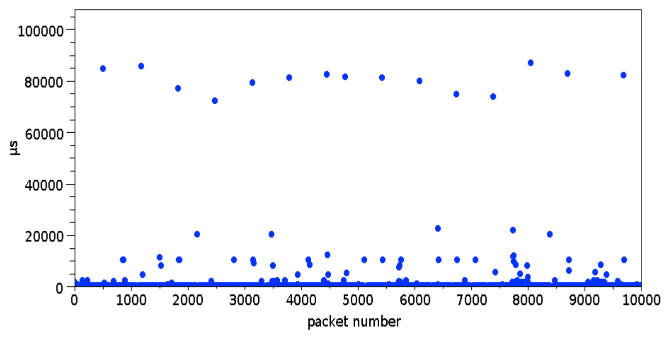


Figure 10. Plot of 10000 Parsing Measurements (60 Bytes)

A. *Comparing Generic and Binary Driver*

To estimate the performance of the XML-based driver a bytecode-driver in plain Java is used to compare both systems. Like the XML-based driver it provides strong robustness and recognizes broken packets. The bytecode driver also uses the same result data type for communication with the sensor gateway. Fig. 11 shows a histogram based on the comparison of the generic and a dedicated bytecode driver parsing 10000 ZephyrHXM packets. Table 2 shows that the datagram parser of the generic driver needs 73% more processing time than the specific binary driver during Datagram parsing. Used in the sensor gateway prototype, the generic driver works reliable. As currently used sensors are manually triggered only a few times a day or sending data with a maximum frequency of 1Hz the additional processing time can be accepted. Measurements of parsing larger datagrams (see. Fig. 9b) show, that having linear complexity, also more complex sensors like an ECG or EEG can be handled.

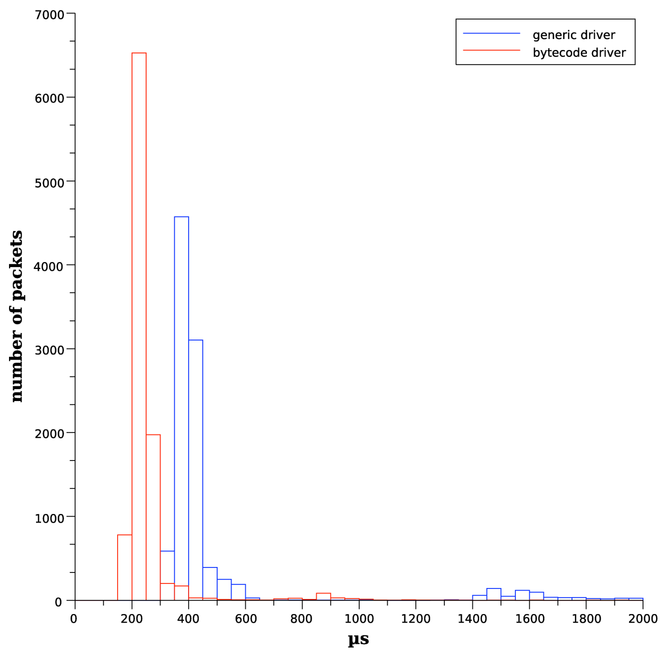


Figure 11. Binary Driver Parsing Time

TABLE II. RESULTS FOR ZEPHYRHXM DATAGRAMS

	Median (10000 measurements)
Binary driver(60 bytes)	214 µs
Generic XML-driver(60 bytes)	371 µs

VI. CONCLUSION AND FUTURE WORK

The sensor gateway prototype currently offers a limited set of events and functions that are implemented in its software. Auto configuration and driver deployment work for common medical sensors measuring glucose, blood pressure and heart rate, which the project currently uses.

The implementation of the auto configuration and driver distribution inside the used Java Enterprise architecture works reliable and does not need much computing power since driver documents are prepared and just have to be looked up in the database and forwarded to the sensor gateway.

The generic sensor driver on the currently used sensor gateway prototype based on a smartphone using Android OS needs more CPU and memory resources than conventional binary drivers that are fitted to the sensor. This resource plus depends on complexity of packet analysis and state machine but is linear to datagram length so it is capable of processing comprehensive measurements made by ECGs for example. In runtime environments using dynamic data types, simple datagram decoding with a generic driver can be similar effective as binary drivers. This is based on the parsing of XML-documents at program startup, generating sensor drivers during runtime. It allows validity checks on datagrams and pushes medical data that is marked with an URN-description to the HMS.

This paper shows that remote configuration and deployment of sensor drivers for Bluetooth sensors is possible without manual configuration of patient's equipment on-site and without using customized binary drivers for each sensor used by the application. After simple sensor types are working well in the architecture the next steps will be to evaluate more complex sensor protocols that need more complex protocol interaction on the sensor gateway and will also need to calculate data of the HMS and send it to the sensors.

ACKNOWLEDGMENT

This work has been supported by the European Funds for Regional Development (EFRE) project "ContextCare" which aims at integrating mobile devices and context-awareness in outpatient care services.

REFERENCES

[1] ContextCare, European fundet EFRE Research Project, http://www.ecs.hs-osnabrueck.de/ContextCare.html, 2009-2011.

[2] R. Hillestad et al., "Can Electronic Medical Record Systems Transform Health Care? Potential Health Benefits, Savings, And Costs," Health Affairs, vol. 24, no. 5, 2005, pp. 1103 -1117.

[3] U. Varshney, Pervasive Healthcare Computing, Boston, MA: Springer US, 2009.

[4] C.J. Su, "Mobile multiagent based, distributed information platform (MADIP) for wide-area e-health monitoring," Computers in Industry, vol. 59, 2008, pp. 55–68.

[5] L.D. Martino, Q. Ni, D. Lin and E. Bertino, "Multi-domain and privacy-aware role based access control in ehealth," Pervasive Computing Technologies for Healthcare, 2008. PervasiveHealth 2008. Second International Conference on, 2008, pp. 131–134.

[6] M. Lipprandt, M. Eichelberg, W. Thronicke, J. Kruger, I. Druke, D. Willemsen, C. Busch, C. Fiehe, E. Zeeb and A. Hein, "OSAMI-D: An open service platform for healthcare monitoring applications," HSI'09. 2nd Conference on Human System Interactions, 2009, pp. 139–145.

[7] P. Kocher, R. Lee, G. McGraw, A. Raghunathan and S. Moderator-Ravi, "Security as a new dimension in embedded system design," Proceedings of the 41st annual Design Automation Conference, 2004

[8] Noueihed, R. Diemer, S. Chakraborty and S. Biala, "Comparing Bluetooth HDP and SPP for Mobile Health Devices," Wearable and Implantable Body Sensor Networks, International Workshop on, Los Alamitos, CA, USA: IEEE Computer Society, 2010

[9] S. Cox, "OGC Implementation Specification 07-022r1: Observations and Measurements-Part 1-Observation schema," Open Geospatial Consortium, 2007.

[10] Specification of the Bluetooth System v2.1 + EDR, Bluetooth SIG, 2007.

[11] M. Baldi and F. Risso, "Using xml for efficient and modular packet processing" Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE, 2006

[12] R.L. Schwartz and P.M. Melliar-Smith, "From state machines to temporal logic: specification methods for protocol standards," Proc. of a conference on The analysis of concurrent systems, New York , USA: Springer-Verlag New York, Inc., 1985, pp. 55–65.

[13] J. Barnett, R. Akolkar, R.J. Auburn, M. Bodell, D.C. Burnett, J. Carter, S. McGlashan, T. Lager, M. Helbing, R. Hosn, T.V. Raman, and K. Reifenrath, State Chart XML (SCXML): State Machine Notation for Control Abstraction, 2010.