Performance Evaluation of a Generic Driver for Medical Bluetooth Sensors

Daniel KÜMPER[1], Ralf TÖNJES[2]
[1]*University of Applied Sciences Osnabrück, P.O. Box 1940, 49009 Osnabrück, Germany*
Email: {d.kuemper, r.toenjes}@hs-osnabrueck.de

Abstract: Outpatient nursing services expend a vast amount of time for manual 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 data. Currently a technical experienced nurse, patient or an additional technical service provider is needed on site to configure new sensors and connect 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. This paper presents a generic sensor driver that is 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. It evaluates its performance compared to a binary driver. The concept is proven by an evaluation of the implementation.

Keywords: Bluetooth, XML, remote/auto configuration, driver deployment

1 Introduction

To support outpatient nursing services, the ContextCare[1] Project is researching ways to use wireless health sensors for supporting documentation and alert services in the context of medical data. The usage of networked medical sensors is a possibility to ensure accurate documentation that requires less time spent by the nurse [1]. A patient-owned device that is able to inquire medical data of various sensors and that also communicates with the nurse's technical equipment is an useful aid. Though wireless sensors offer a high flexibility in their usage, patients as well as nurses may have problems, setting up the sensor network. Because of radio broadcasting of wireless sensors and their accessibility without physical access needed, usually human configuration interaction with those devices is necessary [2]. 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 an interacting computer needs a driver in the form of binary code or interpreter code that has to be deployed before a new sensor device can be used [3]. These drivers can either be distributed with the running software or can be reloaded respectively 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- and 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 to be used must not need on site configuration.
- New binary driver deployments for new types of sensors have to be avoided.

[1] 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.

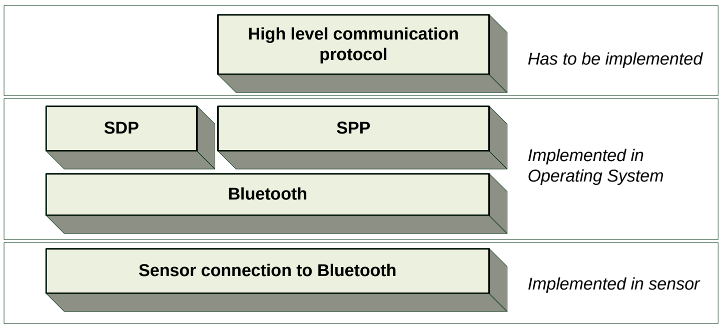

High level communication protocol
Has to be implemented
SDP
SPP
Implemented in Operating System
Bluetooth
Sensor connection to Bluetooth
Implemented in sensor

Figure 1: Communication Stack between Sensor and Receiver

Although a Bluetooth Health Device Profile (HDP) exists [4], most 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. The Bluetooth Service Discovery Protocol (SDP) allows a device to discover services supported by Bluetooth devices and 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 (Figure 1).

2 Architecture

In order to support sensor configuration and driver deployment various Bluetooth sensors use a sensor gateway to connect to a central Healthcare Management Server (HMS) accessible over the Internet. 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 [2]. To allow for a platform independent open interface the 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-documents. Authentication and authorization is ensured by PGP-based message signatures.

3 Remote configuration and secure pairing

In this scenario medical sensors for patients are commonly delivered by a device supplier that is using the centralized architecture for sensor management. The supplier is able to register a sensor to the patient by providing little information. Important data is patient identification, device type, serial number and security credentials

By providing this information, the Management Server can inform a sensor gateway that in the future a new Bluetooth sensor will appear in its Bluetooth neighborhood and is supposed to deliver data. So the sensor gateway receives the data and is able to pair [5] with the Bluetooth sensor device when it notices its first appearance without need for user interaction on site. A XML document that specifies the driver behavior for sensor communication can now be loaded from the management server. A Measurement is initiated either by the Bluetooth sensor or the sensor gateway. This process is visualized in the sequence diagram (Figure 2). It shows an exemplary communication if a new sensor is delivered and registered at the HMS.

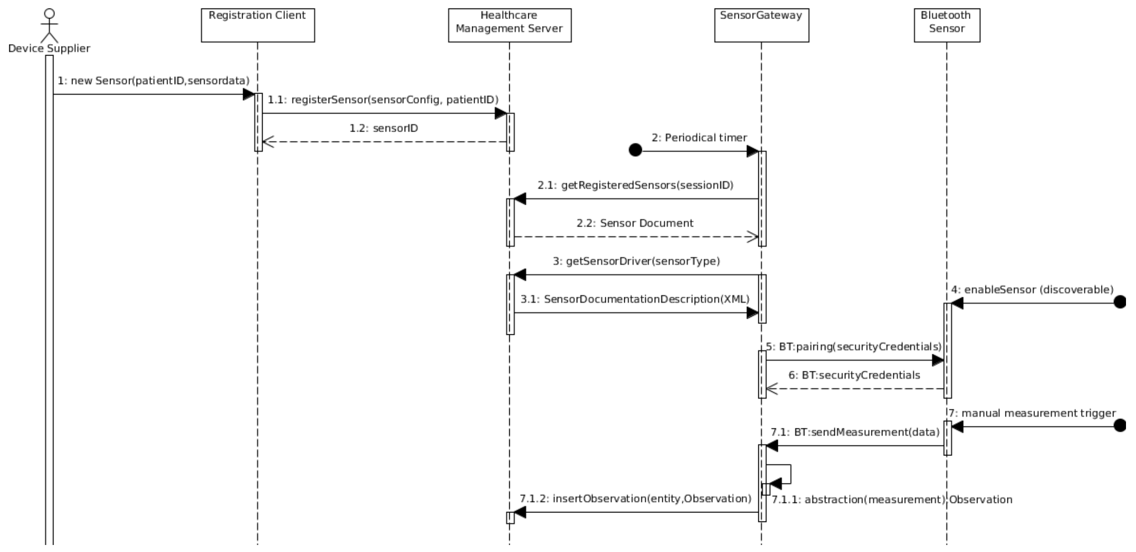

Device Supplier
Registration Client
Healthcare Management Server
SensorGateway
Bluetooth Sensor
1: new Sensor(patientID,sensordata)
1.1: registerSensor(sensorConfig, patientID)
1.2: sensorID
2: Periodical timer
2.1: getRegisteredSensors(sessionID)
2.2: Sensor Document
3: getSensorDriver(sensorType)
3.1: SensorDocumentationDescription(XML)
4: enableSensor (discoverable)
5: BT:pairing(securityCredentials)
6: BT:securityCredentials
7: manual measurement trigger
7.1: BT:sendMeasurement(data)
7.1.2: insertObservation(entity,Observation)
7.1.1: abstraction(measurement):Observation

Figure 2: Sequence Diagram in Case a New Sensor is Added

4 Remote configuration of a generic driver

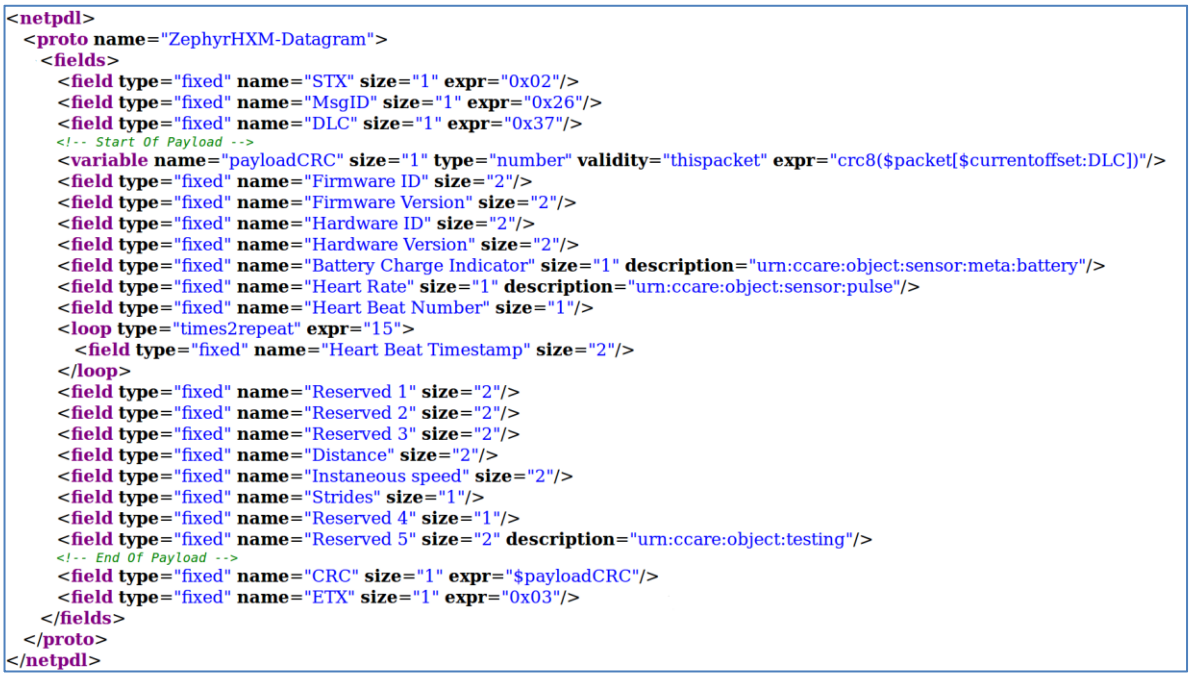
Binary driver deployment for new sensors forces the sensor gateway to download potential security risks after the initial installation of its software. Binary drivers usually have capabilities to access the hardware layer of the sensor gateway. A generic driver for Bluetooth SPP devices which is configured by protocol description documents can minimize 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:

1. The sensor is simply sending information packets without any interaction needed, so only the data packets have to be analyzed.
2. The sensor needs interaction like static *start*/*ack* commands.
3. The sensor needs dynamic interaction where the *master* has to do calculation for the data he sends to the sensor. This could be a CRC over a command or a data lookup.
4. The protocol is controlling a complex 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 XML-schemata (XSD). Furthermore XML-schemata formally describe 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.

4.1 Description of Datagrams

The Network Protocol Description Language (NetPDL) [6] is a simple, 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 (Figure 3).



```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 3: NetPDL-Description of a Heartbeat Sensor (ZephyrHXM)

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* (Uniform Resource Name). Packets that are sent from the sensor gateway to the sensor can also be built out of these documents. Predefined fields with an <*exp*>-element are used to identify the start and end as well as the validity of a packet. On the serial line no other layer or event exist that notices that a new packet is starting or a packet has ended so the decision logic has to use the same layer as the data interpretation. Variables can be declared and the usage of common functions can be described. This feature is needed for protocol fundamentals like a checksum in a datagram.

4.2 State Description

NetPDL does not support the description of a protocol's temporal behavior. Therefore a mechanism like a protocol state machine is needed [7]. This paper employs State Chart XML (SCXML) [8] 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 in SCXML. It is necessary to parse the datagrams for extraction of medical data as an output.

SCXML has 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.

The SCXML Document does not describe 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 send to the Healthcare Management System

Figure 4 shows the simple SCXML based protocol description for a *ZephyrHXM*[2] 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 state machine is described in Figure 5.

```
<scxml version="1.0">
  <datamodel>
    <data name="hxmDatagram">
      <netpdl>
        <proto name="ZephyrHXM-Datagram">
          <!-- Datagram Description in NetPDL (see Figure 3)-->
        </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 4: Example for a SCXML Based Protocol Description

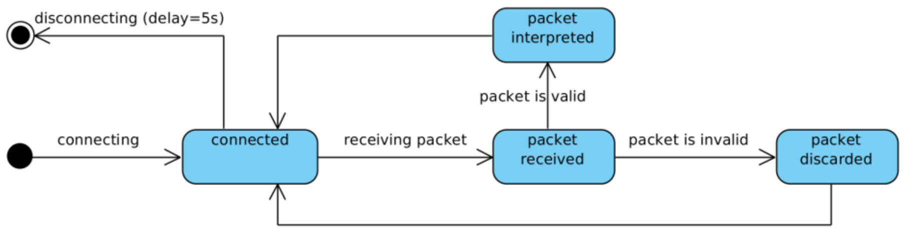


Figure 5: State Chart for Simple Data Protocol

5 Evaluation

The implementation of the generic Bluetooth driver for an Android[3] sensor gateway is used to show its practicability. The 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. Figure 6 shows this process.

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

---

[2] http://www.zephyr-technology.com
[3] Free operating system for mobile devices, http://www.android.com/

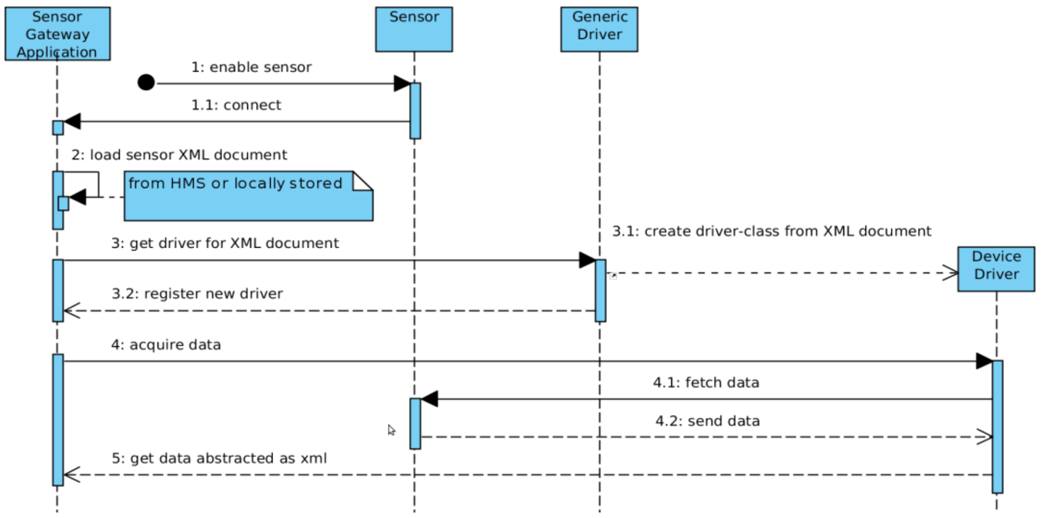
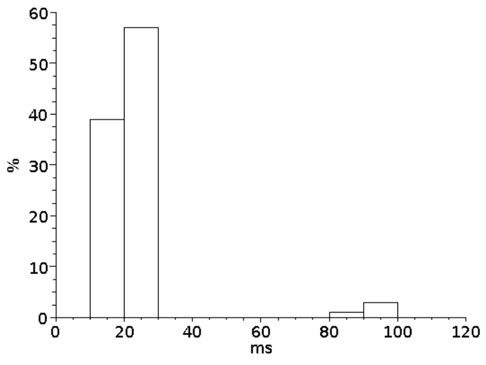

Sensor Gateway Application
Sensor
Generic Driver
Device Driver
1: enable sensor
1.1: connect
2: load sensor XML document
from HMS or locally stored
3: get driver for XML document
3.1: create driver-class from XML document
3.2: register new driver
4: acquire data
4.1: fetch data
4.2: send data
5: get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

Figure 6: Generation and usage of the generic driver

5.1 Performance

To show the performance of the packet processing the datagram description of a ZephyrHXM device is used. It consists of 60 bytes; describing 35 data fields (see Figure 3). 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. 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.

The histogram in Figure 7 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. Figure 8 shows the mean values of 100 driver generations each for the fictional, larger datagrams. This illustrates that the generation-algorithm has a linear time complexity O(n).



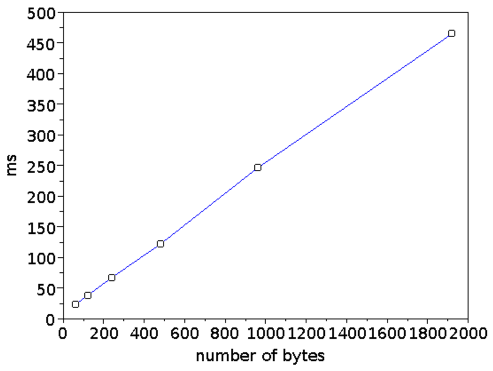


Figure 7: Histogram 60-Bytes Driver Generation

Figure 8: Buildtime for Device Drivers from XML

To determine the sensor data processing speed of the generated drivers, 100000 datagrams of each size are parsed by the corresponding generated driver. Figure 9 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 (<1%) that take the multiple amount of time for processing. These are cut of in Figure 9 for better perception of the processing time distribution. 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 with scattered time peaks is shown in Figure 11. Figure 10 shows the processing time median for the datagram parsers. This illustrates that the parsing-algorithm also has a linear time complexity *O(n)*.

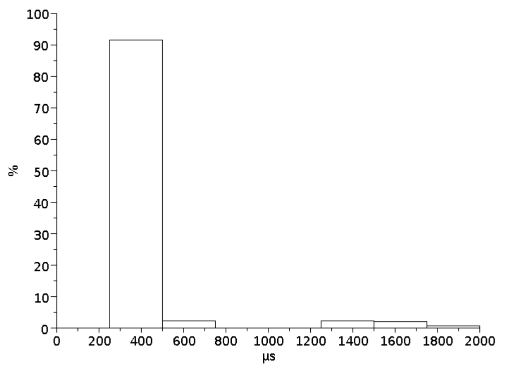


Figure 9: Histogram of Datagram Parsing Time

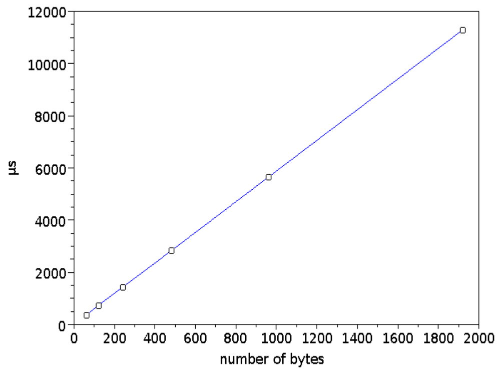


Figure 10: Parsing Time (median)

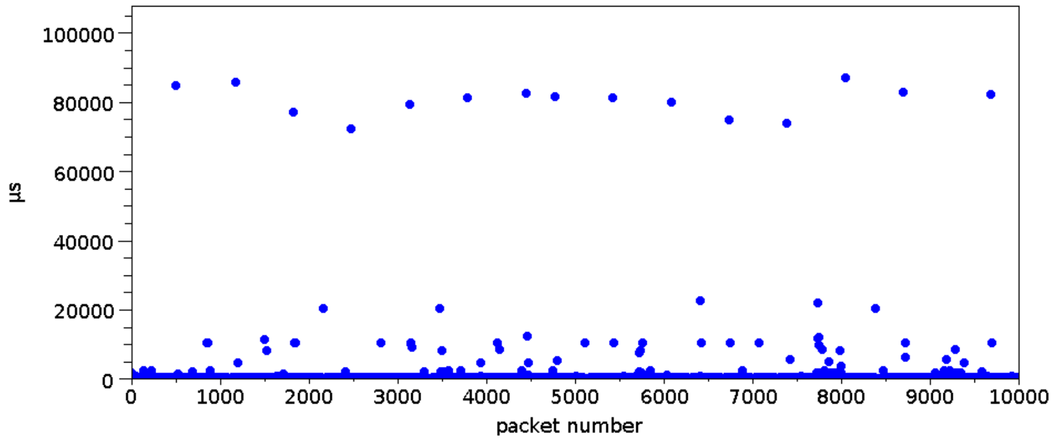


Figure 11: Plot of 10000 Parsing Time Measurements, ZephyrHXM-Driver

5.2 Comparing generic and binary driver

To estimate the performance of the XML-based driver a binary- respectively bytecode-driver in plain Java is used to compare both systems. Like the XML-based driver it recognizes broken packets and it also uses the same result data type for communication with the sensor gateway. Figure 12 shows the histogram based on parsing 10000 ZephyrHXM packets. The measurement also shows scattered time peaks, caused by the System, that are cut off in the Histogram.

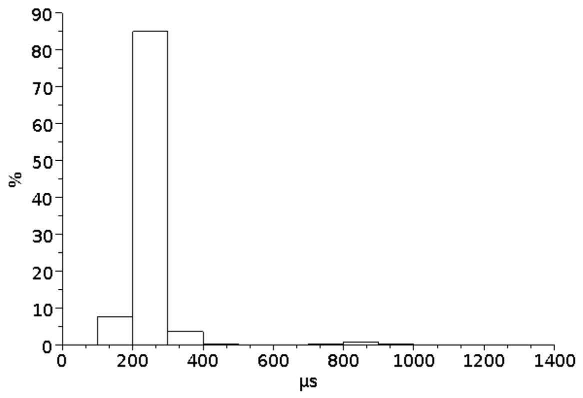


Figure 12: Binary driver parsing time

Table 1: Results for ZephyrHXM datagrams

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

Table 1 reveals the datagram parser of the generic driver needs 73% more processing time than the specific binary driver. 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.

6 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 additional resource consumption depends on complexity of packet analysis and state machine. 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 on-site and without using binary drivers. After simple sensor types are working well in the architecture the next steps will be to evaluate more complex sensor protocols that will need more processing time on the sensor gateway and will also send calculated data of the HMS and send it to the sensors.

7 References

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

[2] D. Kümper, E. Reetz, und R. Tönjes, “Kontextgesteuerte Dokumentation und Kommunikation für Pflegedienste,” *ITG-Fachbericht-Mobilkommunikation*, 2010.

[3] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, und S. Moderator-Ravi, “Security as a new dimension in embedded system design,” *Proceedings of the 41st annual Design Automation Conference,* 2004, S. 753–760.

[4] J. Noueihed, R. Diemer, S. Chakraborty, und S. Biala, “Comparing Bluetooth HDP and SPP for Mobile Health Devices,” *Body Sensor Networks (BSN), 2010 International Conference on*, 2010, S. 222-227.

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

[6] M. Baldi und F. Risso, “Using xml for efficient and modular packet processing,” *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, 2006, S. 6.

[7] R. Schwartz und P. Melliar-Smith, “From State Machines to Temporal Logic: Specification Methods for Protocol Standards,” *Communications, IEEE Transactions on*, vol. 30, 1982, S. 2486-2496.

[8] 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, und K. Reifenrath, *State Chart XML (SCXML): State Machine Notation for Control Abstraction*, 2010.