UML-Based Modeling of Roaming with Bluetooth Devices

Holger Pals
Institute of Computer Engineering
University of Lübeck
Ratzeburger Allee 160
D-23538 Lübeck, Germany
pals@iti.uni-luebeck.de

Zhen Ru Dai
Institute for Telematics
University of Lübeck
Ratzeburger Allee 160
D-23538 Lübeck, Germany
dai@itm.uni-luebeck.de

Jens Grabowski Helmut Neukirchen
Institute for Informatics
University of Göttingen
Lotzestrasse 16-18
D-37083 Göttingen, Germany
{*grabowski,neukirchen*}*@cs.uni-goettingen.de*

Abstract

Bluetooth is a standard for wireless personal area networks. The current standard does not support roaming between different bluetooth networks. In this paper, we will introduce potential roaming techniques for data transfer scenarios using Bluetooth hardware devices and present a corresponding UML model.

Keywords: Bluetooth, Roaming, UML 2.0, Design

1 Introduction

Bluetooth is an established standard for short-range wireless communication. The Bluetooth specification enables small devices to interact within a short range. These device interactions are known as personal area networks [1]. The standards related to Bluetooth include both the hardware (radio, baseband and hardware interface) and basic protocol layers that allow Bluetooth software to run on different Bluetooth enabled devices.

Bluetooth has two types of networks, *piconets* and *scatternets*. A piconet is a network with only one master and many slaves connected to it. Scatternets consist of different piconets connected by devices taking part in more than one piconet. Because there is only one master allowed in each piconet, a Bluetooth device can be master in one and slave in other piconets, or slave in all piconets it is connected to.

The current Bluetooth standard does not support roaming of devices between piconets [2]. If a device is losing the link to its master, no provision is made to transfer the connection to another master. There are several reasons why the Bluetooth standard does not support roaming, including:

- The main usages of Bluetooth are cable replacement of locally fixed devices and support of short-term adhoc networks (e.g. synchronization of desktop PCs and PDAs). For these applications, roaming is not required.
- Bluetooth is supposed to be very simple, efficient and cost-saving. Roaming support would enlarge the Bluetooth protocol stack, making devices more expensive.
- It is hard to define the roaming boundary in piconets because every Bluetooth device can be master or slave.

Nevertheless, roaming within Bluetooth piconets might be useful in some cases, e.g. for Bluetooth-enabled network access using LAN access points. Assuming having more than one Bluetooth LAN access point, roaming might be useful for having a seamless connection even while moving.

In addition to the general usages of Bluetooth, Bluetooth hardware may also be used to establish simple data transfer scenarios that do not make use of the Bluetooth software stack, but only of the lower hardware layers. Because of the standardized hardware interface, the good availability of cheap Bluetooth hardware devices and the usage of the free-of-charge frequency band at 2.4 GHz, Bluetooth hardware is preferable over other (proprietary) radio transmission techniques.

2 The Application

In the context of this paper, the need for a basic roaming support for Bluetooth devices descends from a project between the *Institute of Computer Engineering* at the *University of Lübeck* and several other academic and industrial partners [3]. This project is situated in a medical environment and its goal is to replace the traditional cable-based

monitoring of patients during surgical treatments with a wireless transmission of the patient's monitoring data using Bluetooth hardware devices. By transmitting the sensor data via radio, the mobility of the patient will be increased significantly, the number of artifacts (often caused by the cables themselves) are reduced as well as the overall cost for the replacement of broken cables.

Sensor data like electrocardiogram (ECG), temperature or blood pressure are gathered at a *mobile device*, digitized and transmitted via radio to fixed units (*receivers*). The mobile device is fixed at the patient's bed (or the patient itself) which may be moved during the entire monitoring period. One of the advantages of this wireless monitoring is a continuous data transmission throughout all the different stages the patient passes through (e.g. preparation - anesthesiology - surgery - wake up - intensive care). Thus, the connection between the mobile devices and the receivers mounted at the hospitals walls or ceilings must be handed over from one receiver to the next while the patient is moving. The receivers have to be mounted in such a way that the entire area the mobile device can reach is covered. To allow a seamless connection, the areas covered by the antennas of two adjacent receivers are overlapping (Figure 1).

In this scenario, different units (e.g. sensor units, digitizing unit and radio transmission unit) share the same rechargeable battery pack. The electric power consumption plays an important role in the design of the system. As a consequence, a mobile device only consists of a small embedded device including the Bluetooth chipset and a low-current microcontroller without a complete Bluetooth protocol stack running on it. From now on the term *Bluetooth device* denominates a device using a Bluetooth hardware unit to send and receive data without necessarily using the complete Bluetooth protocol stack.

3 Related Work

As mentioned before, no explicit roaming support is included in the Bluetooth standard. Although roaming support would extend the capabilities of Bluetooth enabled devices decisively, very little work has been done up to now to provide appropriate solutions.

Jozef Steelant introduced different roaming concepts based on packet relaying [4]: He assumes that masters are fixed units connected to a fixed network, while slaves are mobile units. If a mobile unit loses the connection to the fixed unit it currently communicates with, a new unit is chosen to relay all the packets to the communication partner. Steelant's concepts are realized by inserting an extra relaying layer between the Bluetooth *L2CAP* layer[1] and the higher layers that use the L2CAP layer. The insertion of this extra layer allows almost all Bluetooth applications to make use of the roaming capabilities. This kind of roaming is necessary if the communication strongly depends on one special master.

However, in applications where masters are only act as gateways, relaying techniques are not needed for data forwarding, which is the case in our application introduced in Section 2: If the slave loses the connection to its current master, any new master can take over the gatewaying function without having to relay all the received data to the initial master.

In the following, we will introduce a new Bluetooth roaming approach which differs from Steelant's approach. For our approach, we do not need the L2CAP layer of the Bluetooth protocol stacks. We intrude directly on the hardware interface. Also, we do not use the inquiry procedure for roaming, but request directly for a connection to a new master which we calculate by means of preceding spatial positions of the slave. Thus, our approach is more simple and more efficient.

4 Roaming for Bluetooth

For describing our roaming algorithm, we first have to explain the basic procedures for building up connections between Bluetooth devices: In order to establish new connections between Bluetooth devices, the procedures *inquiry* and *paging* are used. The inquiry procedure enables a device to discover which other devices are in its range and what their unique *Bluetooth device addresses* (*BD addresses*) are. Knowing the correct BD address, an actual connection can be established with the paging procedure (*connection request*). Assuming the paged device accepts the connection request, both devices receive the *connection confirmation* message and the connection is established. A device that confirms the connection establishment will automatically become the *master* of the connection while the other device becomes the *slave*. If both devices agree, the roles can be switched any time after completion of the connection procedure.

4.1 Roaming Algorithm

The basic roaming approach presented in this work has been developed in the context of the medical scenario described in Section 2. It assumes that all receivers are connected to a fixed network. This network also connects the receivers to other applications or servers (Figure 1).

The mobile devices are usually moving along the receivers. Because of the piconet structure, the receivers as central units become masters for the Bluetooth connections while the mobile devices become slaves. If a slave runs the risk of losing connection to its actual master, the

[1] The *L2CAP* layer multiplexes logical connections upon physical links.

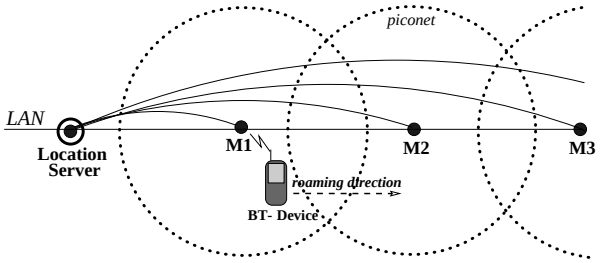

Figure 1. Network Configuration

connection must be handed over to the next master. The slave detects this point by periodically checking the quality of the link to the master. This can be done using the *HCI_Get_Link_Quality* command defined in the Bluetooth standard [2]. If the quality drops below a certain threshold value the next master will be chosen. Because the inquiry procedure is very time-consuming (up to 10s) and no data transmission is possible during this time, the slave tries to connect directly to the next master using paging (consuming below 100ms). This implies that the slave knows to which master it has to contacted to next.

Unfortunately, a slave has no detailed knowledge of its spatial position. For this reason, the information about potential new masters has to be provided by the actual master. To create a suitable list for each slave, the movements of the slaves are tracked by a *Location Server*. The Location Server stores the physical positions of all receivers in form of a connection graph. For each slave that is registered within the network, an ordered list of masters visited before is stored in the Location Server. With this list, a *weak prediction* can be made to determine which master can be connected to in the next future. Weak prediction means in this context that the next master can be predicted with a high probability because the patient's movement is normally directed, e.g. the patient is moving from the preparation room to the operating room going down a certain corridor.

If the slave connects to a master, the Location Server will be informed about this connection by the master. The Location Server then builds up a roaming list of potential new masters, and returns it to the master of the slave. After the reception of this list, the master forwards it to the slave. The order of the roaming list depends on the prediction made by the Location Server, i.e. the list contains all masters that may be reached at the next step starting with the master with the highest probability going down to the masters where the probability is very low. After the connection is set up between the slave and a master, data can be exchanged and the link quality is checked periodically.

The Activity Diagram in Figure 2 shows the activities of a slave necessary for roaming. The slave tries to connect to a master. If the connection is successful, the updated roaming list is transfered to the slave and data can be send. In parallel, the link quality between slave and master is observed. If the quality gets bad, the slave will look in the roaming list for a new master and try to connect to that master directly. If, for any reason, no connection can be established and thus no connection confirmation is received by the slave, a warning message is sent to the user (e.g. by a warning light or a special sound indicating that a problem has occurred). Another warning message is sent to the last master. If the connection to the last master is still alive, the reception of a warning message can be used to initiate appropriate exception handling mechanisms.

Figure 3 shows a sequence diagram for a typical roaming scenario. The diagram consists of four instances: the Location Server, a Slave and two masters Master1 and Master2. The Slave sends data to Master1 and checks the link quality. If the result is below a predefined threshold value (i.e. $\neq$good), a new master will be calculated by Slave by means of its prioritized roaming list. In this scenario, the new master is Master2. Slave sends a connection request to Master2 and gets a confirmation. The confirmation indicates that the connection has been successfully established. Then, Master2 informs the Location Server that it is now the new master of Slave. Master2 receives the new roaming list from the Location Server and forwards it to Slave. Hence, Master2 may exchange data with the slave until the link quality between them becomes weak again.

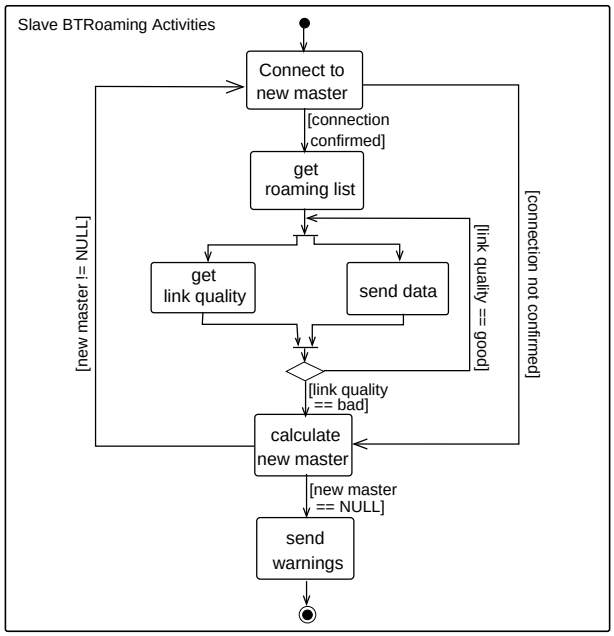

Figure 2. Roaming Algorithm as Activity Diagram

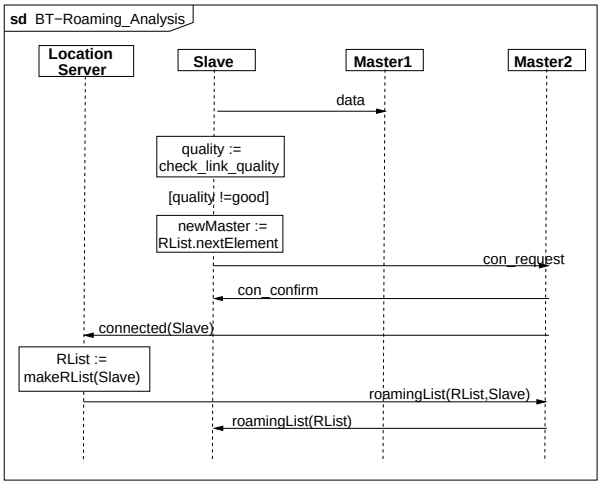

Figure 3. Roaming Analysis

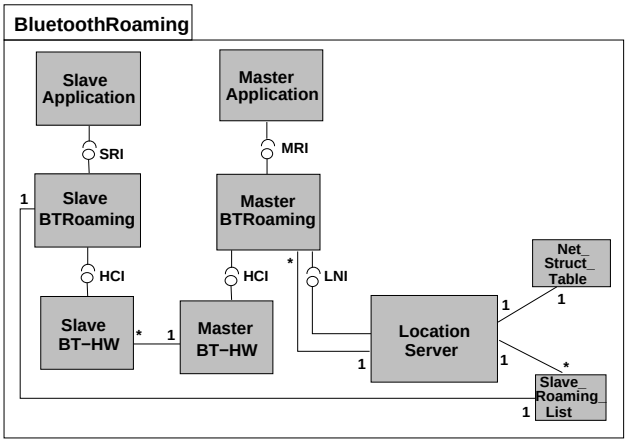

Figure 5. Bluetooth Roaming Package

4.2 Protocol Stack with Roaming Layer

Figure 4 shows the design of the protocol stack resulting from the proposed roaming approach: Special roaming layers (*Slave Roaming Layer* and *Master Roaming Layer*) are added. They take care of the correct transfer of the connections. Our roaming approach makes no use of the higher protocol stacks of Bluetooth. Therefore, the roaming layers are implemented directly on the hardware interface called *Host Controller Interface* (HCI). The application layers are set upon the roaming layers. The interface from roaming layer to application layer is called *Slave Roaming Interface* (SRI) and *Master Roaming Interface* (MRI), respectively.

Additionally, a master is specified as a fixed network node. Thus, it also embodies the LAN protocol stacks to be able to communicate with the local network. The interface between the Master Roaming Layer and the Ethernet is called *Local Network Interface* (LNI).

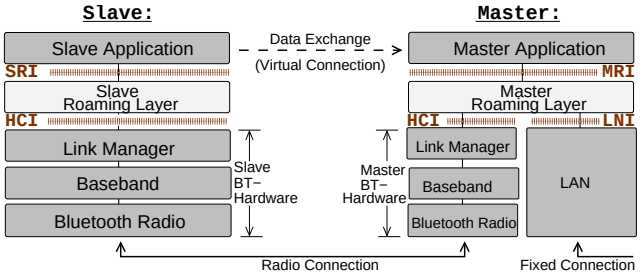

Figure 4. Protocol Stack with Roaming Layer

5 Design with UML

In addition to the Bluetooth roaming algorithm presented in the previous sections, we also investigated the applicability of the newest version (version 2.0) [2] of the Unified Modeling Language (UML) [6, 7, 8] for modeling our roaming approach. The usage of a standardized and widely accepted modeling language like UML has several advantages: It supports the communication among soft- and hardware designers, avoids ambiguities in the description and allows the usage of commercial tools for documentation, analysis, implementation and testing during system development. In this Section, we describe an architectural view on our Bluetooth roaming scenario by means of a UML package diagram, show the communication among the UML Bluetooth classes in form of sequence diagrams and present the local behavior of a slave by using a UML state machine.

Figure 5 shows an UML package diagram with different classes involved in our Bluetooth roaming approach. Similarity can be recognized between the classes in Figure 5 and the Bluetooth protocol stacks in Figure 4.

The slave classes are called Slave Application, Slave BTRoaming and Slave BT-HW (Bluetooth Hardware). The interfaces SRI and HCI connect the class components with each other.

A Slave BT-HW is connected to one Master BT-HW. Similar to the slave classes and interfaces, there are the classes Master Application, Master BTRoaming and Master BT-HW and the interfaces MRI and HCI on the master's side.

Master BTRoaming class is connected to the Location Server, which represents a node in the local network, by means of the interface LNI. Location Server owns a Net_Struct_Table and

[2] UML 2.0 has been adopted by the OMG in June 2003. Currently, it is at its standard finalization stage. In this paper, we follow the approach of *U2 Partners* consortium [5], who is the main submitter of UML 2.0.

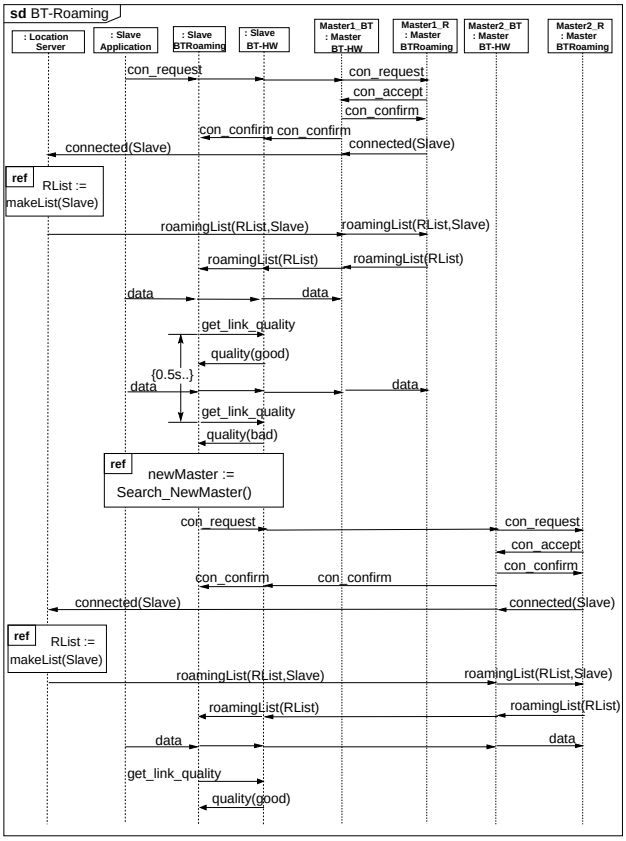

Figure 6. Roaming Scenario Design

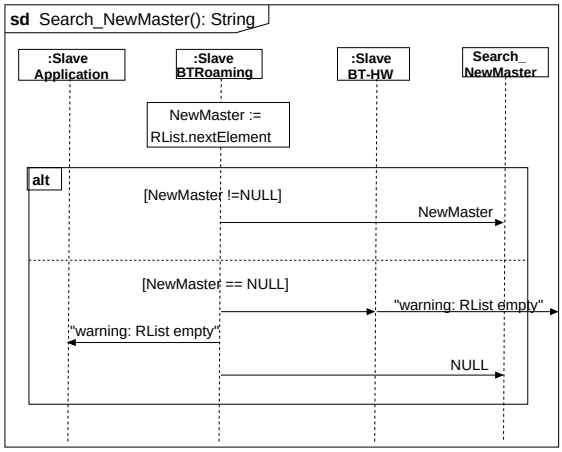

Figure 7. *Search_NewMaster()* **Function**

several Slave_Roaming_Lists. There is exactly one roaming list for each slave. The Net_Struct_Table is a static table which provides information about the structure of the local network and the physical position of the masters as necessary for calculating each Slave_Roaming_List. In contrast, the instances of Slave_Roaming_List are changing dynamically. The Slave_Roaming_List is updated by the Location Server whenever a slave roams to a new master. Since a copy of each updated Slave_Roaming_List is transferred to its slave there is also a one-to-one association between Slave_Roaming_List and Slave BTRoaming.

In Figure 6, the sequence diagram depicts a detailed roaming scenario. There are eight different instances in the diagram: One location server instance called Location Server, three slave instances named Slave Application, Slave BTRoaming, Slave BT-HW, and four masters instances with BT-HW and BTRoaming instances for each of Master1 and Master2.[3]

The scenario starts with a connection request from the application instance of the Slave to Master1.[4] The hardware instance Master1_BT confirms the connection establishment and the roaming instance Master1_R informs the Location Server that Slave is now under its responsibility.[5]

Hence, the Location Server calculates and updates the roaming list RList of the Slave and sends it to Master1_R. Master1_R forwards the RList immediately to the Slave Roaming instance. Henceforward, data can be exchanged between Slave and Master1 until the link quality becomes bad.

The verification of the link quality is done periodically between the Slave BTRoaming instance and the Slave BT-HW Instance. If the link quality is proved to be bad, a new master is needed. For that, the function Search_NewMaster() is called by Slave BTRoaming. This function looks up in the RList and picks out the name of the best reachable neighbouring master and returns the name of the new Master to Slave BTRoaming (Figure 7). In case that RList is empty, a warning will be sent to both the Slave Application and the old Master (if it is still possible). Additionally, a NULL value is returned to instance Slave BTRoaming.

In our scenario (in Figure 6), the new Master is Master2. Thus, a connection request will be sent from Slave Roaming instance to Master2_R instance. If Master2 BT-HW confirms a successful connection establishment, the Location Server will again be informed about the new status of Slave. It updates the roaming list and sends it to the roaming instance of the new master. Master2_R forwards the list to Slave BTRoaming and data exchanges can be started.

A set of scenarios, like the ones presented in Figures 5, 6 and 7, can be analyzed and used to generate local views of class behaviors. One possibility of UML to describe such local behaviors are state machines.

As an example, Figure 8 shows a state machine of the

[3]The application instances of Master1 and Master2 are not shown because roaming is independent from the application layers.

[4]In order to provide an intuitive understanding of the signals, we abstracted the names from the real Bluetooth signal names in the Bluetooth specification [2].

[5]Since there is a lot forwarding traffic between the instances, we only describe the source and the destination instances of a message.

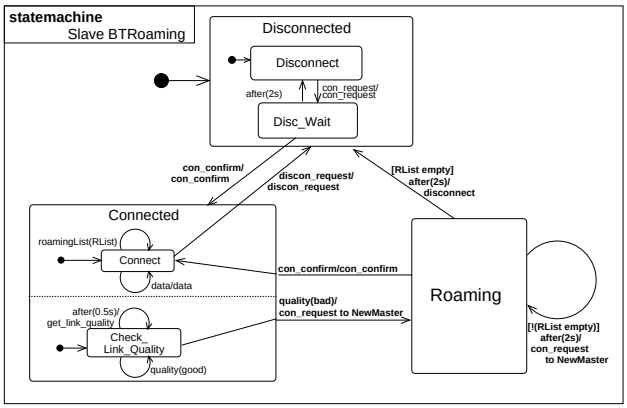

Figure 8. Statemachine of class Slave BTRoaming

slave roaming instance Slave BTRoaming. This instance receives messages from the Slave Application instance and the slave hardware (Slave BT-HW) instance. The diagram contains the states Disconnected, Connected and Roaming. Disconnected is a composite state with multiple sub-states, Connected has orthogonal sub-states.

In the beginning, the Slave BTRoaming instance is in the sub-state Disconnect of state Disconnected. If it receives a connection request from the application instance, it forwards the request to the hardware instance and goes into the sub-state Disc_Wait, waiting for a connection confirmation from the hardware. If the confirmation message does not arrive within 2 seconds, Slave BTRoaming instance goes back to the Disconnect state. If the confirmation is received by the roaming instance within time limit, the slave is Connected and goes into the sub-state Connect. In this state, data can be received from the application instance and forwarded to the hardware instance (state Connect). In parallel, link quality can be verified (state Check_Link_Quality). The Roaming state will be reached, if the link quality becomes bad. Herein, a new master is picked out and a connection between the slave and the new master will be established. From state Roaming, the roaming instance can either get Connected to a new Master or be Disconnected again, if the roaming list has been exhaustively searched and no master can be found.

Even though the newest version of UML is still under development, we got the impression that UML is very well suited to model roaming for Bluetooth devices. The different kinds of diagrams force us to describe the roaming from different perspectives and on different levels of abstraction. We believe that UML improved the modeling process and helped to avoid ambiguities in the description.

6 Summary and Outlook

In this paper, we have introduced roaming techniques for data transmission using Bluetooth hardware devices and have analyzed them by UML diagrams. The roaming techniques we introduced in this paper are suitable for Bluetooth piconets. Another roaming domain for Bluetooth is to explore roaming within a scatternet. Further investigations concerning the direct connection procedure using the proposed roaming lists should be done in order to analyze its efficiency in contrast to the usage of the basic inquiry procedure.

Due to missing tool support for the UML version 2.0, we were not able to analyze our model automatically. Our future work will include such a validation. We also plan to investigate the possibilities to generate executable code for Bluetooth roaming from our UML model. Experience with earlier versions of UML have shown that at least VHDL and C skeletons [9] can be generated automatically from UML descriptions.

In another direction of work [10], we will show how to specify tests for the designed UML model. For that, we use the concepts of the newly adopted UML 2.0 Testing Profile by OMG [11].

References

[1] http://www-106.ibm.com/developerworks/wireless/library/wi-handblue.

[2] *Specification of the Bluetooth System (version 1.1)*, Bluetooth Special Interest Group, http://www.bluetooth.com.

[3] http://www.iti.uni-luebeck.de/Research/MUC/EKG/.

[4] J. Steelant, *Roaming for Bluetooth*, Haute Ecole Valaisanne, Sion, Switzerland, 2001, http://www.holtmann.org/lecture/bluetooth/.

[5] http://www.u2-partners.org/.

[6] http://www.omg.org/uml.

[7] *UML 2.0 Superstructure*, Draft Adopted Specification at OMG (ptc/03-07-06), July 2003.

[8] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[9] D. Harel and E. Gery, "Executable object modeling with statecharts," *IEEE Computer*, vol. 30, no. 7, pp. 31–42, 1997.

[10] Z. R. Dai, H. Pals, J. Grabowski, and H. Neukirchen, *UML-Based Testing of Roaming with Bluetooth Devices*, First Hangzhou-Lübeck Conference on Software Engineering (HL-SE'03), 2003.

[11] I. Schieferdecker, Z. R. Dai, J. Grabowski, and A. Rennoch, "The UML 2.0 Testing Profile and its Relation to TTCN-3," Testing of Communicating Systems – 15th IFIP International Conference, TestCom2003, Sophia Antipolis, France, LNCS 2644, Springer, May 2003.