CERO: CE RObots Community

Peter Ibach, Nikola Milanovic, Jan Richling, Vladimir Stantchev, Andre Wiesner, and Miroslaw Malek

Institut fuer Informatik

Humboldt University of Berlin

{ibach|milanovi|richling|vstantch|wiesner|malek}@informatik.hu-berlin.de

Abstract. We describe our experience creating a robots community using standard hardware and software – the Windows CE operating system, XScale development boards, and LEGO Mindstorm actuators. We focus on an open and extendable distributed robotics system, which in contrast to most existing proprietary robotics systems, supports cost-effective integration and assembly of components while providing dependable communication and coordination. To demonstrate feasibility we have developed the CERO Framework for ad hoc cooperation in robots communities employing a service-oriented architecture with the main building blocks: ad hoc networking for different wireless technologies (e.g., Bluetooth, WLAN, or 433 MHz radio communication), a protocol for interconnecting development board and actuators, and a CE.NET based software framework supporting complex movements, consensus protocols, cooperation, and positioning. It offers high-level interfaces for programming complex tasks in cooperative robotics. We also present an overview of intended applications.

1 Introduction

Today's robotics hardware is expensive and far away from being commercial-of-the-shelf (COTS). The same applies to software; most existing robotics systems implement proprietary interfaces, use specific operating systems, and require particular development, monitoring, and testing tools. Both facts imply that it is expensive, difficult, and sometimes impossible to integrate robotics systems and standard COTS components – particularly when considering distributed robotics systems that depend on reliable and timely communication and coordination. However, assembling and interconnecting COTS components to build flexible, inexpensive but dependable (wireless) networks of robotics systems would be highly desirable. This gives rise to the question of how far we can go using standard hardware, operating systems, communications protocols, and tools to build such robotics systems. Ultimately, our aim is to provide a community forming software framework for quick design and realization of cooperative robotics systems out of COTS components.

Looking for inexpensive, flexible, and composable robotics systems, several products targeted for the gaming market seem to be quite suitable for concept demonstration, e.g., Fischertechnik Computing

or LEGO Mindstorm. However, those systems typically offer limited computing capabilities, software development possibilities, and connectivity. Thus, our idea was to combine LEGO Mindstorm actuators with Windows CE compatible hardware. CE runs on Pocket PCs, but as well on development boards that are embeddable and available at low prices compared to proprietary robotics systems. Microsoft's .NET Compact Framework is integrated within CE and offers a comprehensive environment for embedded systems programming using standard technologies, based on standard hardware, and supported by an extensive toolset for development, monitoring, and testing.

To handle complexity, present robotics systems usually either do not consider distributed coordination and focus on stand-alone, special-purpose robots, or, if cooperation is needed, the overall system has to be designed such that possible situations, appropriate actions, and the appropriate allocation of tasks are known in advance. Example of the first case is a robot diver for rescue operations on wrecked ships; example of the second case is an assembly line in a car factory. So a key human ability – to cooperate ad hoc and thus adapt to dynamic environments and unforeseeable conditions – is still not an every day option in the world of robotics. Therefore we introduce ad hoc networking in robots communication and cooperation, thus enabling dynamic and adaptive interaction.

Our idea of ad hoc cooperation within a robotic community is based on our past research activities – consensus and consensus protocols. We describe the results of this work as well as a method of obtaining flexible components that perform community forming processes in CERO. The concept allows to use relatively simple and inexpensive robots for a variety of actions in different environments that, until now, have been performed by expensive special-purpose robots. This approach is very flexible and moreover, it can eliminate a single-point-of-failure problem – in a community of robots, tasks can be distributed and, if one robot fails, its tasks can be carried out by another robot.

2 Robots Communication

Wireless communication among the robots is obligatory to support mobility. Since a network of mobile robots may continuously change its topology, ad hoc networking is essential to cope with dynamics.

In ad hoc networking all nodes may be used as intermediate relays when sending messages. Therefore, they must posses computing capabilities to maintain message relaying and routing. In an ad hoc network, routing is performed on demand, that is, when a node that has to forward a message does not know where to send it next, triggered by message arrival it discovers the next hop. Thus, ad hoc networking can quickly adapt to dynamic topology changes. On the other hand it requires additional bandwidth for coordination messages, causes computation overhead for on demand route discovery, and may result in high communication latency. Thus, message end-to-end timing in ad hoc networks is hardly predictable.

However, cooperative tasks under real-time constraints need to be dependable and communication has to guarantee message delivery times. Therefore, we use augmentation to support specific quality of service requirements in ad hoc networking. The specific issues that we address in CERO are: the use of location awareness and partitioning detection to (1) minimize the additional processor and bandwidth consumption for ad hoc routing and to (2) guarantee message delivery within real-time constraints.

2.1 Combining Network Flooding and Position-based Routing

There are two basic ways to perform ad hoc routing: network flooding and position based routing. Network flooding is done by flooding the network with control packets that try to locate the destination node, or the first node that has an active route to the destination node. A survey of network flooding algorithms can be found in [10].

Naive network flooding exhaustively consumes bandwidth, processor capacity, and energy. We proposed an algorithm that employs clustering of mobile nodes in order to minimize the number of control messages that are flood into the network [9]. The approach keeps computing capabilities needed to perform the clustering reasonably low on the considered CE boards. The minimization of the number of control messages also helps to establish real-time guarantees for message delivery.

If nodes are aware of their coordinates, position-based routing can be performed (usually using GPS). Algorithms with guaranteed message delivery exist for such setups [11]. However, position sensing is difficult to realize in indoor environment where GPS does not work.

Since we would like CERO to be applicable both indoor and outdoor we are investigating flexible solutions that utilize position sensing services with respect to availability and specific service parameters such as positioning accuracy [2]. Outdoor positioning then will likely be performed using GPS, if available, while WLAN, Bluetooth, or RFID-based position sensing, if available, might be more appropriate when inside a building. Once the position is known to the robots, our partitioning prediction scheme [1] can be employed in order to support cooperative robot tasks. When a current movement pattern of one or more nodes is detected that might lead to network partitioning, an alarm is triggered. Thus, appropriate movement corrections can be invoked to keep the network fully connected and prevent the cooperative tasks from failing because of udeliverable messages.

2.2 Binding CERO with Existing Wireless Technologies

The protocol is designed to be independent of the physical communication; nevertheless the actually employed physical layer has certain impact on non-functional properties and the way that some protocol details are implemented. Currently, we are experimenting with WLAN, Bluetooth, and 433MHz radio and testing various properties such as range, connection stability, timing behavior, and energy consumption.

Blutooth supports master-slave communication, where one node (master) synchronizes and controls up to 8 slave nodes. This is a welcomed addition to our clustering scheme since nodes do not have to spend additional time on leader election and synchronization. However, during our tests Bluetooth delivered some questionable performance, especially concerning range and connection stability. In contrast to Bluetooth, 433MHz radio communication was very dependable, the range could be easily configured and connections were stable. The downside of this technology is that all primitives needed for clustering have to be implemented additionally on the CE board. The main disadvantage of WLAN is its energy consumption, however this technology is widely spread and well supported. The same problem as mentioned for 433MHz board arises here: the clustering algorithm must be additionally implemented since WLAN also does not support native clustering. We expect that with more mature Bluetooth technology it will be the best choice, but alternatives such as 433MHz radio communication or WLAN are possible as well.

3 Interconnecting Robots and Control Boards

The basic idea of CERO is the connection of COTS robotics hardware and embedded control boards to form a distributed robotics system that delivers high functionality and connectivity at low costs.

An advantage of that concept is the ability to exchange components: If a robot lacks computation or communication abilities it is possible to use another control board but the same robot. On the other hand, we can easily change the robot that is controlled by the control board. This enables, e.g., testing a rescue robot with simple and primitive hardware in the laboratory and later replace this hardware by the actual rescue robot using the same control board and software.

In order to reach this goal we consider interoperability and composability at both hardware and software level and define an appropriate protocol.

3.1 Hardware

At hardware level we asume that the robot has a very limited controller that is able to execute simple commands (read a value from a sensor, write a value to an actuator), run a simple protocol engine and to support serial communication (RS 232) at low speed (e.g., 2400 bit/s). These requirements are fulfilled by many low-cost controllers such as the AVR family from Atmel or by robotic systems from the toy market such as LEGO Mindstorm [3]. Therefore, we use standard RS 232 communication to link the robot and the control board.

3.2 Software

To manage the interconnection via RS232 the following steps are involved (Windows CE 4.2): After opening a serial port with `CreateFile`, we use `GetCommState` and `SetCommState` for configuring port

settings, `WriteFile` and `ReadFile` for communicating trough the port and `CloseHandle` for closing the serial port. We manage communication events using the `WaitCommEvent` and `SetCommMask` functions.

Communication at the higher layers involves the Real-time Communications (RTC) API which relies on the Session Initation Protocol (SIP), the Real-Time Transport Protocol (RTP) and the PSTN/Internet (PINT) internetworking services. Furthermore, we use standard API functions for management of related data structures both at the level of the Win32 API and the CE.NET API.

3.3 Protocol

The communication process is initiated with a Wakeup-call from the control board. Once both participants have expressed a need for communication, robot and control board need to agree on functional and nonfunctional parameters like number and type of sensors and actuators as well as the level of fault tolerance. If a decision has been made, a continuous flow of data has to be guaranteed. If the established connection is closed the robot will cease all activities and wait for another Wakeup-call.

After the initial Wakeup-handshake the control board determines in a three step procedure if the robot meets its requirements. In the first step the available sensors, actuators and fault tolerance will be negotiated.

For this protocol, only the granularity of the sensors is needed (i.e. number of bits), semantic issues are of little to no importance. The control board sends a vector of required sensor datatypes and (optional) minimum and maximum time intervals between sensor readings to the robot. The latter answers with a vector of the same length (with respect to number of elements), returning one of the states "OK", "No sensor connected", "Not sensitive enough" or "Can't be delivered in time". If evaluation shows that requirements and available ressources don't match, the control board either needs to cut down on requirements and retry negotiations or it simply quits them. In order to be able to restart negotiation, maximum and minimum requirements need to be defined. After an acknowledge in form of a vector containing the timing and datatypes agreed upon, the process is repeated for the available actuators with their respective datatypes. Here, we only define the time interval between control messages, either relative to sensor input (which defaults to three sensor inputs per output) or give a specific time span. The latter may force a reevaluation of sensor timings.

The process for negotiating fault tolerance is relatively simple (in terms of the protocol). The control board sends a request regarding numbers of bit errors to be detected and corrected (corresponding to statistical failure rates of the data channel). The robot has a predefined amount of methods for error correction implemented and automatically selects the method requiring a minimum of overhead while still fulfilling the request. It then returns the ID of the selected method. If the robot is not capable of handling the requested type of error management, it reports so. The control board then needs to (re)evaluate if

downgrading it's requirements is an option or not and communicate this decision to the robot. With an acknowledgement from the robot the setup-sequence has been completed.

To ensure continuous operation, there has to be an ongoing exchange of data packets. There are basically two ways of doing this, depending on the chosen strategy of either pushing or polling. We decided to do both.

In case of simple pushing as strategy the sending of "alive"-messages is not necessary, data will be transmitted instead. The robot cycles through the sensors and transmits the values in regular intervals, which makes it predictable and easy to handle. The control board has defined timeslots at which it sends control messages to the outputs. This scenario's drawback is that sensor values have different priorities for the user and need to be available at specific times. As a solution, one can also exchange timing requests during setup sequence using a few more bits per [port:datatype] tuple. Ideally, the programmer defines maximum and minimum time span between arrival of data per sensor and the robot has to check whether it finds a schedule that fits those requirements.

Simple polling of data would have been less expensive in terms of data transmission and data handling on the control board. Automaticallly generated "alive"-messages would still have been necessary in case there were no data request in the message queue and the control board would be responsible for the scheduling during runtime, which may take time it does not have. Also, if the new data do not arrive on time, the currently available data may be too outdated to work with, which violates our requirement of being responsive. Using polling alone, during regular operation we would not have to deal with data we don't need, but also are not able to act "on time" in times or trouble.

In our implementation, sensor data requests out of order can still be answered by using the timeslot specified for control messages, which doesn't effect the sensor loop. In case of changing requirements during runtime (like a huge increase in control messages needed for transmission of sensor data), one reinitiates the setup sequence or, even better, just the part of it doing the scheduling.

4 Community Formation

The approaches discussed in Section 2 (ad-hoc communication) and in Section 3 (interfacing a simple robot with an inexpensive control board) enable us to build inexpensive robots that are able to communicate in an ad-hoc manner. More specific, they are able to cooperate in groups forming communities.

This section discusses this issue in detail starting with our previous experiences on the areas of consensus in Section 4.1 and community building in Section 4.2 and then putting everything together in Section 4.3.

4.1 Consensus

Consensus as defined in [4] allows a set of fault-free processor elements to get the correct value of a part of the system state while ensuring that all other fault-free processor elements get the same result independent from actions and values of faulty processor elements. This does not imply that all elements start with the same value, it only implies that all fault-free elements finally come to the same result using a consensus protocol.

Our research in the area of consensus protocols with the goal to improve responsiveness has resulted in novel protocols and architectures [7, 5].

A system implementing COnsensus for REsponsiveness was built: CORE [7]. It was used to develop the applications "Unstoppable Orchestra" and "Unstoppable Robots" [8] which demonstrate responsive behavior even in presence of Byzantine faults. Within "Unstoppable Orchestra" a set of computers plays music in a way that each computer plays one instrument. If one of the computers fails, its part is taken over by another computer without stopping the music. "Unstoppable Robots" is a simulation as well as implementation of several robots balancing an unstable plate. These robots are controlled by distributed controllers in a way that each controller may control one or more than one robots. If controllers are faulty (or even if they try to destroy the balance of the plate which is Byzantine behavior) the consensus protocol ensures balance of the plate as long as enough fault-free controllers are available.

4.2 Group Communication

Our previous experience in group communication helped in forming directions for CERO development. We have recently completed the PLASMA project – Person Loss Avoidance System for Mobile Applications [12]. The goal of the project was to create a system that offers supervision of the mobile group of persons, primarily children, when they are playing in the kindergarten, visiting a museum or an amusement park, or when they are on excursion.

Since the requirement was high mobility, we solved the problem by tagging every child (mobile node) with receiver/transmitter, creating a group out of all nodes and spanning a wireless ad hoc network among all group members. That way the system is not dependent on external infrastructure and can be set up dynamically and virtually everywhere. All group members periodically report to one supervisor node (a teacher) who is carrying a CE.NET handheld. If one or more group members are missing, an alarm is triggered on the device and the supervisor has instant information on who is missing and where he/she was last seen.

The aspects of the PLASMA project that we transferred into CERO are: ad hoc communication between group members, security and real time behavior. Ad hoc communication is an obvious requirement since we would like to support high mobility while being independent of environment, weather, buildings,

etc. In PLASMA ad hoc communication is also used to limit the allowed group diameter by defining maximum number of hops that a message may travel over before the sending node is pronounced out of the group. The other parameter we used was signal strength. By combining these two values we were able to fine tune the maximum distance that a node is allowed to move from the group. Faced with security issues, we decided not to encrypt all communication, as a performance tradeoff. Since tags have very limited resources, it would simply take too much computing time and energy to encrypt all data before sending. That means that anybody could 'listen' to the communication between nodes. However, all senders were authenticated, and no intruder could broadcast wrong information since it would be just discarded by other nodes. In the CERO setup there is no such limitation: our hardware is strong enough to both encrypt communication and perform authentication. We are still investigating feasibility of full communication encryption and its aspects on the overall real-time behavior of the system. In the PLASMA project we were able to guarantee deadline in which all group members would report to the supervising node and be detected either as present or absent. This time scales linear with the number of group members. In the CERO setup such estimate is much more difficult because it depends on the nature of the cooperative task that the robots are currently performing.

4.3 Interoperability of CERO Components

While the PLASMA project provided valuable results regarding ad hoc communication, security, reliability, and real-time behavior for CERO, it did not address higher level objectives of the robots community – knowledge exchange, evolutional learning, as well as planning and scheduling of cooperative tasks.

A unified format for storage and exchange of information is a prerequisite for the CERO community. This ensures that every community member can communicate with anybody within the community - e.g. robots, PDAs, servers. We decided to use a SQL Server based infrastructure for these tasks. It offers the benefits of using our existing infrastructure and applications for data analysis and security, a build-in functionality for automatic data synchronization in SQL Server for CE and also allows us to use existing applications for data access on all platforms.

There are four basic types of information we can store into the CERO database - location data, robot status data, robot task data and cooperative task data.

Another important aspect of community forming in a distributed environment such as CERO is the development of a common programming framework for it. The CERO Framework is a CE.NET based software framework with several .NET extensions for servers and supports complex movements, consensus protocols, cooperation within communities and positioning. As part of our vision of NOMADS [6] CERO supports properties such as mobility, dependability and adaptivity. By offering a common programming framework CERO allows the definition and implementation of various cooperative applications while

providing important non-functional properties at the architectural level. The application developer is insulated from the underlying communication, consensus and positioning protocols and can define the application using the higher level APIs of the CERO Framework.

5 Applications

5.1 Swarm robotics

Let us first consider applications where robots only have rudimentary capabilities: They are autonomous (i.e., they act independently of any remote control), have limited memory (i.e., each robot knows its current status but does not remember where it has moved and what it did before), and have very limited self-control capabilities (i.e., they only have a small set of simple rules that guide their behavior). A group of such robots, even though each individual is only able to perform at the level of reflex actions, may exhibit complex behavior at the corporate level. Typically this very soon converges to some static state, oscillates in a fixed periodicity, or results in some unpredictable chaos. However, under certain conditions, some degree of "intelligence" may turn up. Such intelligent behavior is also referred to as swarm intelligence and biological analogies to insect communities (e.g., an ants colony) are inspected. To further investigate the conditions under which swarm intelligence does emerge, simple but flexible swarm robotics applications that can be easily used for various experiments are highly demanded in teaching and research.

Example applications are: cyclic stop-and-go with obstacle avoidance, follow the leader, pheromone-based path finding, clustering around food, tagging (catch a robot and tag it), manhunt (team tagging), cooperative object transportation, balancing robots.

Thus, to support easy set up of experiments examining swarm robotics, CERO provides means for:

- Detection and identification of robots nearby
- Exchange of sensor data and other status information
- Consensus and voting mechanisms in case of differing or contradicting values
- Further methods for fault detection and prevention such as plausibility check and noise filtering
- Rule-based programming supporting: `if <sensor status condition> then perform <action>`

5.2 Human-machine interaction

As already mentioned above, community forming can emerge from rudimentary communication among robots. But it will get more complex – and as well appealing – if humans and human behavior come into play. Human-to-machine interaction which can be opposed to machine-to-machine interaction requires intuitive user interfaces for control and monitoring purposes. In the extreme case robots mimic human

behavior to tell their status including emotions. The CERO Framework will provide means to express emotions on an abstract level, i.e., emotions such as pleasure, fear, or anger which will be translated to hardware specific movements of limps or other actuators. Robots need to sense and understand such behavioral patterns. Finally, robots that convincingly mimic and sense human-like behavior will be able – to some degree – to take part in human communities. A blinking LED, as a trivial example, may show the robots heartbeat where the beat frequency relates to the robots exertion/excitement. Further examples are the handshaking humanoid ASIMO, the PeopleBot, which has enhanced cognitive capabilities, or the wagging AIBO RoboDogs. CERO will support such human-to-machine interactions simplifying experiments that further investigate the way humans and robots can interact.

5.3 Gaming

We envision gaming as a very promising area for the ideas and technologies we developed during the CERO project. Particularly interesting for cooperative robots communities are robotic simulations of team sports. Here we will be able to implement additional features when compared to existing solutions (e.g., Robocup, Sumobots). Such features include an overall team strategy, creation of specialized robots (e.g., defensive and offensive players). We find that similar features can also be useful in strategic games where players will command armies of robots and pursue a complex strategy to defeat each other.

We plan to evaluate possibilities of using some of our solutions within CERO in the areas of mobile and online gaming. Possible applications range from ad-hoc building of gaming communities from mobile users to consensus protocols and related technologies for synchronization of graphics, movements, and other information in distributed gaming scenarios.

5.4 Property protection

Applications in the area of property protection are well designated for CERO as cooperation of multiple entities is a clear added value in such scenarios. A task such as patrolling robots can have several cooperative benefits - areas covered by every robot can be designed with minimum overlaps, they can also be modified on demand (e.g. when a robot fails its area is covered by its neighbours) and when a robot faces some intruder it can call other robots for reinforcement. We see similar benefits in the area of field exploration too - covered areas can be dynamically assigned and robots and can call neighboring robots to help them.

5.5 Rescue action

Robots for rescue actions are typically made as custom and high-cost specialized hardware. They can only perform one given task for which they were designed. Our vision, on the other hand, is to have many

small multi-purpose robots, such as provided by the CERO. Using community principles discussed so far, one can build a group of robots of various capabilities (motion detectors, cameras, image recognition, temperature detection) and send them to investigate affected by disaster sites. Although none of them is capable to perform such a complex task alone, together as a group they might prove to be very efficient. The main objective is to eliminate a single point of failure. If one motion detector robot fails, other robots with the same capability will take over its assignment. The price and the efficiency of the solution are also important, since rescue actions can be mounted quickly and less expensively with groups of multi-purpose robots than a single specialized machine.

The other application of the same idea would be field exploration, such as sending a group of multi-purpose robots to explore a cave or other site that is not easily accessible or is dangerous to humans. By communicating ad hoc and adapting to the environment conditions, robots would be deployed independently of any control center, perform the given task and return and report to the human observer.

5.6 Smart house

The area of smart house and overall facility management is also a suitable marketplace for cooperative robots. Possible tasks include cleaning where relatively simple robots can be employed to reduce overall costs and thus offer more complex tasks. Moving is also a possible area of usage, as well as specialized tasks such as maintenance of building facilities - power and communication lines, elevators, lightning,etc. Visitor guidance is another area where we can use cooperative robots. One benefit here when compared with other electronic solutions, such as PDA or voice guidance, is the ability to also facilitate the movement of the visitor so he does not get lost or walks through areas which are not designed for visitors.

5.7 Moving routers

Community forming principles could also be used to form a dynamic and mobile support for wireless communication. Suppose that we equip each CERO robot with a wireless access point and create a group of such robots with the task to support movement of users in some predefined area. Robots would thus act as a moving routers and provide infrastructure for wireless communication by reinforcing critical links that are about to fail or to run out of bandwidth.

6 Conclusion and Future Work

CERO demonstrates the viability of our idea to combine standard development boards, operating systems, and software components with inexpensive, highly configurable robotics hardware and organize the composite entities in a community using ad hoc networking, group communication, and consensus. CERO

has a wide range of possible applications in areas such as swarm robotics, human-machine interaction, gaming, property protection, rescue action, smart house, and moving routers. The implementation of case studies in these areas is one of the objectives of our future work. We will further focus on improvements and optimizations of the communication between development board and actuators, as well as improvements of the communication and consensus protocols among the community members. We also consider CERO's distributed, service-oriented architecture as particularly well suited for researching and testing our approaches and models in the area of architectural translucency.

Microsoft Research awarded the CERO project an Innovation Excellence Award for Embedded Systems. This work is supported by this grant.

References

1. Milic B., Milanovic N., and Malek M. Prediction of Partitioning in Location-aware Mobile Ad Hoc Networks. In *Proceedings of the Hawaii International Conference on System Sciences, HICSS-38, (Minitrack on Quality of Service in Mobile and Wireless Networks) (to appear)*, Hawaii, USA, 2005.
2. P. K. Ibach and M. Horbank. Highly available location-based services in mobile environments. In *International Service Availability Symposium 2004*, Munich, Germany, May 2004.
3. LEGO. *LEGO Mindstorm*. http://mindstorms.lego.com/eng/default.asp, 2004.
4. Barborak M., Malek M., and Dahbura A. The Consensus Problem in Fault-Tolerant Computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.
5. Malek M. Omniscience, Consensus, Autonomy: Three Tempting Roads to Responsiveness. In *Proceedings 14th Symposium on Reliable Distributed Systems*, Bad Neuenahr, Germany, September 1995. Humboldt University of Berlin.
6. Malek M. Introduction to NOMADS. In *ACM Computing Frontiers 2004*, Ischia, Italy, 2004.
7. Werner M. *Responsivität – ein konsensbasierter Ansatz (Responiveness – A Consensus-based Approach), PhD-Thesis*. WBI GmbH, 2000.
8. Werner M., Polze A., and Malek M. The Unstoppable Orchestra: A Responsive Distributed Application. In *Proceedings of the 3rd International Conference on Configurable Distributed Systems (ICCDS'96)*, pages 154–160, 1996.
9. Milanovic N., Radovanovic A., Cukanovic B., Beric A., and Tesovic N. Bluetooth Ad-hoc Sensor Network. In *Proceedings of the IX International Telecommunications Forum TELFOR 2001*, Belgrade, Serbia, 2001.
10. Milanovic N., Malek M., Davidson A., and Milutinovic V. Routing and Security in Mobile Ad Hoc Networks. *IEEE Computer*, 37(2):61–65, February 2004.
11. Bose P., Morin P., Stojmenovic I., and Urrutia J. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. *Wireless Networks 7, pp. 609-616*, 2001.
12. Buhle S., Hensel M., Kreiser D., Zapotoczky J., and Milanovic N. Person Lost Avoidance System for Mobile Applications, CSIDC 2004 World Finals Report. *http://www.informatik.hu-berlin.de/~milanovi/humboldt.pdf*, 2004.