Distributed Software Management in Sensor Networks using Profiling Techniques

Gerhard Fuchs, Sébastien Truchat, Falko Dressler
University of Erlangen-Nuremberg
Autonomic Networking Group, Department of Computer Science 7
Martensstr. 3, 91058 Erlangen, Germany
{gerhard.fuchs,sebastien.truchat,dressler}@informatik.uni-erlangen.de

***Abstract* –Methodologies for efficient software management in wireless sensor networks (WSN) need to be investigated for operating and maintaining large-scale sensor networks. Until now, some network-based approaches have been described that are limited in terms of scalability, i.e. dependency on reliable end-to-end communication, and security. In this paper, we describe a distributed software management architecture using profiling techniques. We exploit the advantages of robot-based reconfiguration and re-programming methods for efficient and secure software management. The developed methods are depicted in detail. Additionally, demonstrate their applicability and advantages.**

***Keywords* – wireless sensor network, software management, profiling techniques, re-configuration and re-programming**

I. INTRODUCTION

Software management for wireless sensor networks is an ongoing research area. Due to the heterogeneity of employed hardware platforms and the low resources in terms of processing power, available memory, and networking capacities, new approaches for efficient software engineering are needed. An overview to the issues in sensor nodes is provided in [2]. Culler and coworkers describe the necessity for network-oriented software architectures. Issues on the questions of how to configure, re-configure, program, and re-program networked embedded systems such as sensor nodes are discussed in [7].

The development and the control of self-organizing, self-configuring, self-healing, self-managing, and adaptive communication systems and networks are primary research aspects of our Autonomic Networking group at the chair for Computer Networks and Communication Systems. In the frame of the ROSES (Robot Assisted Sensor Networks) project, we study these aspects on a combination of mobile robots and stationary sensor networks. We call this combination a mobile sensor/actuator network. In this context, we distinguish between sensor assisted teams of mobile robots and robot assisted sensor networks. An example for the former scenario is sensor-based localization and navigation. We developed a robot control system named Robrain [8] for general purpose applications in multi-robot systems. Part of this work was an interface between the robot systems and our sensor motes (see below). This allowed us to study the applicability of the ad hoc sensor network for localization assistance [3]. An example for the latter scenario is assistance for maintenance and deployment of sensor nodes as well as for task and resource allocation [4]. Currently, we are investigating methods for adaptive re-configuration of sensor nodes using mobile robot systems. Two separate goals should be achieved using these techniques: calibration of sensor hardware and re-programming based on changes in the environment. In order to address these issues, we apply profiling mechanisms as described in this paper.

The use of mobile robots for reconfiguring single sensor nodes and, therefore, larger ad hoc sensor networks has many advantages. For example, the robot systems usually have much more available resources and can store and maintain software modules needed by the sensor nodes. Additionally, there are multiple reasons for employing mobile robot systems for re-programming the sensor nodes First of all, applications like sensor calibration can be done only locally. Calibration means to have an expensive high quality sensor attached to the robot system and much cheaper sensors distributed in the field. We discovered that these sensors need a re-calibration in regular intervals. A second reason is security. To achieve mutual trust between a sensor node and a server is much easier if less complex communication protocols are used. Therefore, sensors should deny any reconfiguration from distant systems. Finally, the localized reconfiguration using mobile robots can speed up the re-programming task and save resources like bandwidth and energy because the sensor network is not influenced by the re-programming activities.

Similar work was done mainly based on network-centric reprogramming. For example, the Deluge system [1] was developed for re-programming Mica2 motes. Deluge propagates software update over the ad hoc network and can switch between several images to run on the sensor nodes. An role assignment system was developed at the ETH Zurich [5] to switch between multiple tasks depending on the current requirements. The flexible exchange of software components in TinyOS was investigated at the University of Stuttgart. The developed toolkit FlexCup [6] introduces software engineering methods for sensor node programming. Incremental network (re-)programming was studied in [9]. The primary focus of this work was on the delivery of software images over an ad hoc network.

The rest of the paper is organized as follows. In section II, the developed profiling techniques are depicted. The proposed reconfiguration scheme is discussed in detail in section III. Implementation details are shown in section IV. Some conclusions summarize the paper.

II. PROFILING

Our profiling (or profile matching) concept has grown in the frame of a project about interoperative systems: Mo.S.I.S (Modular Software engineering for Interoperative Systems). One essential goal was to elaborate a generic reconfiguration mechanism based on profiling in combination with a lightweight non-blocking RPC mechanism [12].

This profiling mechanism consists of two parts:

1. A definition of profiles that characterize a software service, e.g. software modules, and such profiles that characterize environments, i.e. platforms on which services can be offered, e.g. sensor nodes.
2. A definition of profile matching rules defining how these platforms can be reconfigured with these services. The word reconfiguration stands here in general for any new software configuration (in the sense of loading new software).

An ontology can be seen as a formal specification on how to represent objects or entities, and the defining of rules on how they stand in relationship. Therefore, profiling can be seen as a kind of interoperative reconfiguration ontology.

Composite Capabilities / Preference Profiles (CC/PP) [10] offers a way to describe profiles. One typical application for CC/PP is content adaptation. A client sends a HTTP request including its profile, and the web server matches the document profile with the device profile to adapt the document that is sent back in the HTTP response.

This scenario offers similarities with the re-configuration and re-programming issues as investigated in the ROSES project:

1. A sensor node first sends its profile to a robot system (a profile characterizing its hardware capabilities and the installed software modules).
2. In a second step, the application for the sensor node has to be compiled by assembling several software modules stored as code fragments on the mobile robot that performs the role of a local server. The selection of these software modules is made based on a profile comparison between the hardware profile of the sensor node and the profile of the software module.
3. At the end of the reconfiguration, the profile characterizing the installed software on the node has to be updated.

For adaptive sensor network re-programming, we had to define a byte-oriented profile in order to meet the demands of the very limited hardware and communication resources. An ID (identification number) defines each hardware element plugged into a node. Such an ID can be stored in one byte. In the same way, an ID stored in one byte defines each software module. Usually, no more than three hardware elements are installed at a single sensor node. Also, we decided that a node can host up to five software modules. Therefore, we consequently need only eight bytes to characterize the hardware and software of a single sensor node. As we only need a few RPC commands, we reserve only one byte for the RPC command. Thus, the communication datagram for reconfiguration only needs to be nine bytes.

The robot, having definitely more resources, hosts the database where the complete profile description related to an ID can be found. In that way, once the node has transmitted its profile, the robot can decide with which software modules to reconfigure the node through profile matching.

III. RECONFIGURATION

A. Application Scenario

In our laboratory, we use the Robertino robot platform developed at the Fraunhofer Institute AIS[1] running Embedded Linux and the Mica2 sensor motes running TinyOS developed at the University of Berkeley[2]. We have connected a MIB510 programming and serial interface board with the Robertino and installed a Mica2 node as a base station. This enables our robot to communicate directly with the wireless sensor network. In the following, we concentrate on the reprogramming of the sensor nodes for dynamic adaptation to environmental changes.

For re-programming, we prepare our sensor motes with an initial binary, which contains a module for profiling concerns. The robot can use this module to receive information about the hardware configuration and the currently installed applications of the sensor mote, e.g. Mica2 / Mica2dot, temperature measurement / localization. On the robot, we store nesC-code and code templates that are described by profiles. This enables the robot to select and adapt the source code concerning the current context and requirements and, finally, to create a new binary for the sensor node. The robot can install the image over the air.

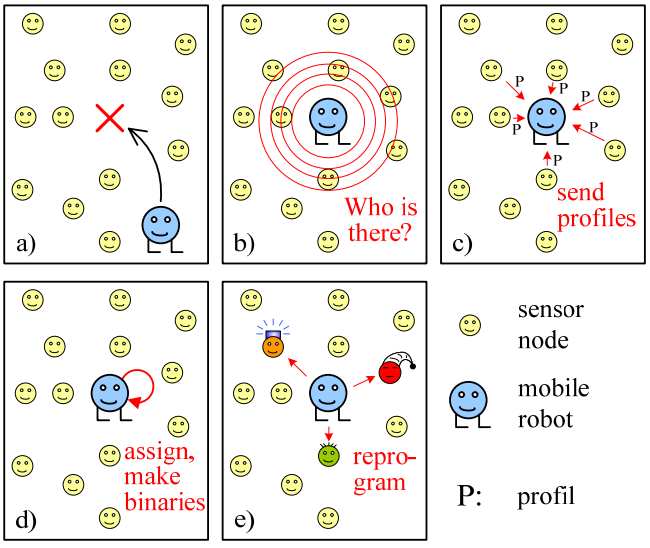

Fig 1. Application scenario for reconfiguration

[1] www.openrobertino.org
[2] www.tinyos.net

Fig 1 shows the principal concept of reconfiguration:

a) Depending on the goal, the robot drives to the position in the sensor network, where reconfiguration is necessary (we do not assume a particular navigation scheme, various mobility models can be applied).
b) The robot collects information about the environment, builds the context and explores its neighborhood. In this step, additional actions can be initiated such as preparing the sensor calibration or starting an algorithm for dynamic addressing schemes.
c) All sensor motes, which have received the exploration message, send their current profiles that contain information about the hardware and software of the node.
d) The robot uses the information gathered in steps b) and c) to assign the roles of the sensor motes, optimized for the current goal. As a result, it creates the new binaries of the sensor motes. Eventually, additional processing or communication with other entities might be necessary unrelated to the re-configuration itself.
e) The robot re-programs selected sensor motes over the air.

B. *Formal Description*

The activity diagram of the reconfiguration process of the mobile sensor network is shown in Fig 2. We distinguish between strategic and technical actions. The strategic actions are responsible for the behavior of the whole system. They depend on a global goal (e.g. a task) and control the reconfiguration process of the sensor network. The technical actions are independent of the goal. They are always the same and provide the functional basics for reconfiguration. Without them, no autonomous reconfiguration is possible.

In the rest of the paper, we use the following shortcuts:

- <XYZ> is short for the action / activity with the name XYZ
- NP = node profile
- AP = application profile
- MP = module profile
- NP*/AP*/MP* = at least one NP/AP/MP

After the start of the reconfiguration process, <prepare> is started. The robot does some initial actions, which depend on the goal, e.g. it moves to a particular position. Then, it determines the current context, i.e. requests the profiles of neighboring sensor motes. By the use of this information, it works out the configuration and the list of applications that is needed in the current context. The results parameterize the technical actions <match profiles> and <make binaries>.

Actions and activities:

a) First, <explore> is started. As a result, the robot has the current configuration of all sensor motes in its sphere of influence in form of a list of NPs.
b) <match profiles> determines all possible combinations of applications and modules, which can run on the nodes. For this, the MPs and APs on the robot and the NPs from the nods are needed. The output is a list of matching profiles. Each entry has the form (NP, AP*, MP*).
a) <assign> reduces the cardinality of the result. This action is a strategic action and, therefore, depends on the global goal. The output is the final mapping for the reconfiguration of the sensor nodes.
b) In <make binary> the binaries for the sensor nodes are generated. This action needs the list of matching profiles from the previous step, the code templates, the source code, and the configuration. The result is a list of (node address, binary)-mappings.
c) Finally, this is taken as the input for <reprogram>, which is the last step of the reconfiguration loop. The robot re-programs the notes over the air.

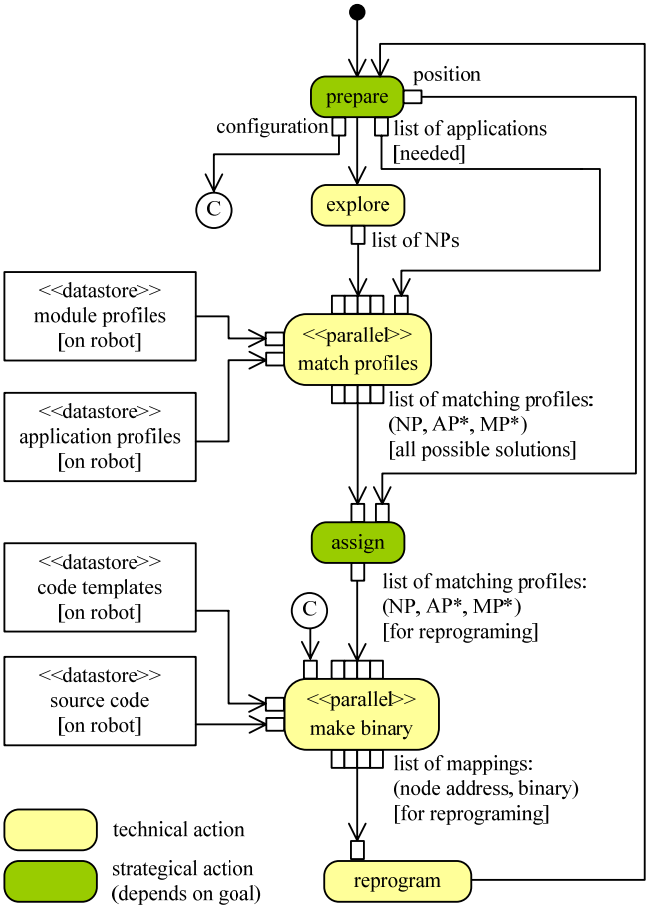

Fig 2. Activity diagram of the reconfiguration process (NP = node profile, AP = application profile, MP = module profile; XP* = at least one profile of X)

IV. Realization of the Technical Aspects of Reconfiguration

In the following subsections, some details on the realization of the profiling concept for wireless sensor networks are discussed in more detail.

A. Neighborhood Exploration

Neighborhood exploration covers two separate steps. The first one, which is optional depending on the configuration of the wireless sensor networks, is the setup of address information. In several scenarios, addresses are not necessary to operate a sensor network. Therefore, also the communication between the mobile robot and a given sensor node cannot be directed using address information. We propose to initiate a dynamic addressing algorithm first [11]. In the lab, we implemented a simple addressing scheme for this initial task.

The second step is to explore the neighborhood. All nearby sensor nodes must be identified and their profiles must be collected. A simple broadcast to the neighboring nodes allows to send a single request to all relevant sensor nodes. Then, each node sends a reply including its current profile. This profile is taken as input for the following profile-matching algorithm.

B. Profile Matching

The primary goal of profile matching is to create all possible combinations of executable source code. Again, we use a straightforward terminology for the definitions. (NP, AP*, MP*) means on the node described with NP the applications described by AP* with the modules described by MP* can be installed. Each module or application can be realized using different source files. For example, a module may consist of various sub-modules that can be found in multiple nesC files.

For profile matching, the name of the description in the profile is important for its realization. In the following, we present several examples for profiles of nodes, applications, and modules. The profiles are depicted in pseudo-code of the profiles. Usually, the node profile is only a bitmap. Please note the importance of having unique names of modules and applications.

In these examples, NP1 is a typical Mica2 sensor mote that has installed additional senor hardware. The node is used for light measurement. AP1 is an application that measures the temperature. It was developed for Mica2 motes. MP1 and MP2 represent alternatives of a software module for different hardware systems. Finally, MP3 is a hardware-independent module to calculate some statistics of measured data.

```
node {
   properties:
      address  = 1;
      board    = mica2;
      sensors  = mts310;
      appl.    = LightMeasurement;
}                                                    (NP1)
```

```
application {
   properties:
      name     = TemperatureMeasurement;
      modules  = TempSensorM, CalcM;

   requirements:
      board    = mica2;
}                                                    (AP1)
```

```
module {
   properties:
      name  = TempSensorM;

   requirements:
      sensor   = mts310;
}                                                    (MP1)
```

```
module {
   properties:
      name  = TempSensorM;

   requirements:
      sensor   = mts101;
}                                                    (MP2)
```

```
module {
   properties:
      name  = CalcM;
}                                                    (MP3)
```

An application can be installed if the AP matches the NP (board-property) and all MP listed in AP.modules match NP (sensor-property). Profiles can be extended at any time. Each module profile describes a code fragment. This is either a static nesC-file or a configurable template. If the profiles match, the described code fragments for the APs and MPs can be compiled. The complete profile matching procedure is depicted in Fig 3.

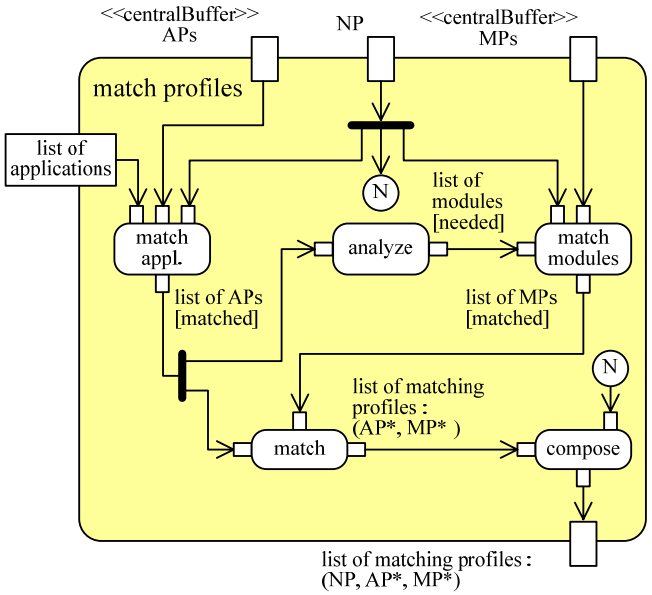

Fig 3. Activity diagram of the profile matching action (NP = node profile, AP = application profile, MP = module profile; XP* = at least one profile of X)

<match appl.>, <match modules>, and <match> use rules working with the profiles.

First, <match appl> is initiated. In our example, the operation "<match appl>: NP1.board == AP1.board → match" is successful, i.e. the application can be complied for the given sensor hardware. Afterwards, <analyze> is employed to generate a list of needed modules. In our case, TempSensorM and CalcM are involved and forwarded to the <match modules> procedure.

The second part of the profile-matching algorithm is the module match. In the provided example, the <match modules> operation performs the following checks: "NP1.sensor == MP1.senor → OK", "NP1.sensor != MP2.senor → !OK", and "MP3 → OK (no requirements)". A list of MPs is created that meet the requirements.

Finally, <match> perfoms a test of the APs and MPs. Using the example again, "<match> → (AP1, MP1, MP3)" produced a final list of matching profiles that build the basis for composing the matching profiles. If <match> produces no match, i.e. the empty set, no modules to build the desired application are available and no corresponding binary can be generated. At <match> (AP*, MP*) is one entry of the list. Finally, "<compose> → (NP1, AP1, MP1, MP3)" is called to add the node profile to the profile list for further processing during the binary generation.

C. Generation of the Binary

To be flexible, the robot builds the binaries of the sensor motes just in time. Therefore, it needs a dynamic source code selection and generation system.

Fig 4 shows the activity diagram for making one binary. One static input pin belongs to the code templates for the generation of the wiring, the node profiles and the configuration, another to the source code of the modules (nesC–files). The dynamic inputs are the current configuration and the matching profiles. The goal is to create a binary, that runs on the node described by NP and contains all applications and modules described by AP* and MP*.

<split> extracts the information of the profiles and provides it for further processing. <select src> selects the source code, which is described by the APs and MPs (there is a unique mapping) and puts it into a temporary buffer. <generate wiring, node profile and configurable modules> generates the dynamic nesC-files, depending on the current configuration and the different combinations of APs and MPs, and puts them into another temporary puffer. Therefore, the code templates are used. <compile> compiles all the nesC-files. <compose> maps the resulting binary with the corresponding node address.

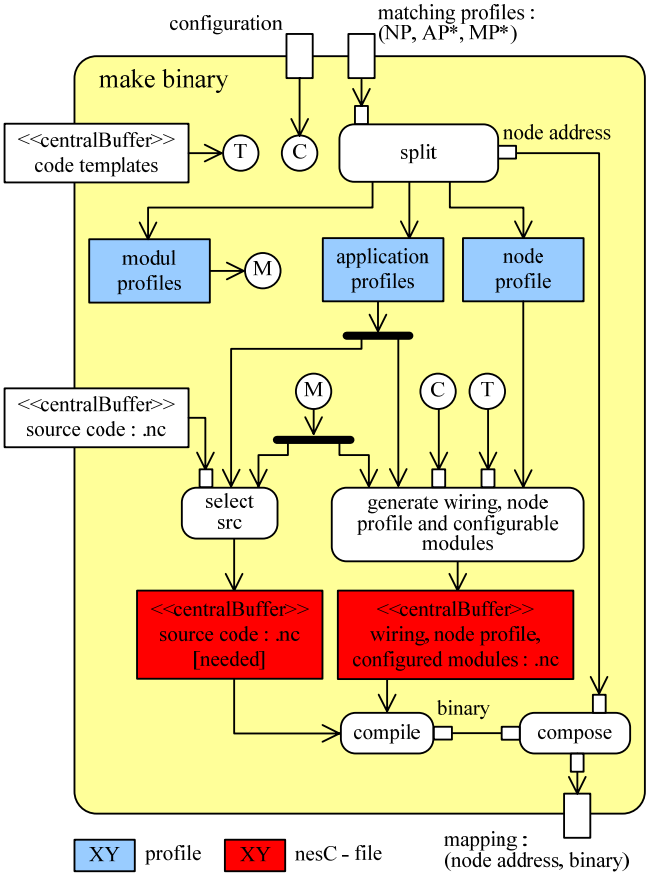

Fig 4. Activity diagram of the make binary action (NP = node profile, AP = application profile, MP = module profile; XP* = at least one profile of X)

The structure of TinyOS programs requires some additional handling in combination with the selection of

source files. First, the wiring between the modules must be defined. Using the profile-based description, the APs, MPs, and templates can be used for an unambiguous wiring. Secondly, some parts of the nesC-code have to be adapted to different hardware configurations. To prevent the necessity of providing mostly identical software modules, i.e. such that differ only in few lines of code, we propose the utilization of templates. A template and a configuration defined by a profile will be substituted to a configurable software module that is adapted to a particular hardware configuration.

In a final step, the node profile is transformed to a nesC-file that can be compiled to a new binary. This binary reflects the application profile and corresponds to the actual hardware capabilities.

D. Reprogramming

The last part of the node reconfiguration using profiling techniques is the re-programming of the nodes for which new binaries have been generated in the last step. We intend to use an extended version of Deluge for this purpose. In the context of code generation, the re-programming method needs to be considered in terms of special modules or software modifications required for re-programming purposes. In our case, an additional software module will always be installed in any generated binary image that is responsible for network-centric re-programming.

V. Conclusions and Future Work

In this paper, we presented a method for distributed software management in wireless sensor networks using profiling techniques. We elaborated the profile matching architecture and presented the necessary steps for node reconfiguration. The scenario is based on stationary sensor networks and mobile robots that perform management and configuration tasks. Based on the available resources at the robot systems, sophisticates software architectures can be maintained and applied for task allocation, sensor calibration, and general-purpose reconfiguration of surrounding sensor nodes. Additionally, the complexity of communication over the ad hoc network as well as the security concerns in network-based node re-programming are minimized.

The presented profiling techniques build the basis for the development of dynamic reconfiguration in large-scale sensor networks. The adaptive exchange of software modules depending on the global goals and environmental factors has become possible. In future and related work, strategies for the robot-based re-programming must be developed that are optimized for efficiency and coverage.

References

[1] A. Chlipala, J. Hui, and G. Tolle, "Deluge: Data Dissemination for Network Reprogramming at Scale," 2004. (http://www.cs.berkeley.edu/~jwhui/research/)

[2] D. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo, "A Network-Centric Approach to Embedded Software for Tiny Devices," Proceedings of First International Workshop on Embedded Software (EMSOFT 2001), Tahoe City, CA, USA, October 2001.

[3] F. Dressler, "Sensor-Based Localization-Assistance for Mobile Nodes," Proceedings of 4. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, Zurich, Switzerland, March 2005, pp. 102-106.

[4] F. Dressler and G. Fuchs, "Energy-aware Operation and Task Allocation of Autonomous Robots," Proceedings of 5th IEEE International Workshop on Robot Motion and Control (IEEE RoMoCo'05), Dymaczewo, Poland, June 2005, pp. 163-168.

[5] C. Frank and K. Römer, "Algorithms for Generic Role Assignment in Wireless Sensor Networks," Proceedings of 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys), San Diego, CA, USA, November 2005.

[6] M. Gauger, "Dynamic Component Exchange in TinyOS (Dynamischer Austausch von Komponenten in TinyOS)," Master's Thesis (Diplomarbeit), Distributed Systems, University of Stuttgart, April 2005.

[7] V. Handziski, J. Polastrey, J.-H. Hauer, C. Sharpy, A. Wolisz, and D. Cullery, "Flexible Hardware Abstraction for Wireless Sensor Networks," Proceedings of 2nd European Workshop on Wireless Sensor Networks (EWSN 2005), Istanbul, Turkey, February 2005.

[8] M. Ipek and F. Dressler, "An Extensible System Architecture for Cooperative Mobile Robots," Proceedings of IEEE International Conference on Robotics and Automation (ICRA 2006), Orlando, Florida, 2006. (submitted)

[9] J. Jeong and D. Culler, "Incremental Network Programming for Wireless Sensors," Proceedings of First IEEE International Conference on Sensor and Ad hoc Communications and Networks (IEEE SECON), June 2004.

[10] M. Nilsson, J. Hjelm, and H. Ohto, "Composite Capabilities/Preference Profiles: Requirements and Architecture," W3C, W3C Working Draft 21, July 2000. (http://www.w3.org/TR/2000/WD-CCPP-ra-20000721/)

[11] Y. Sun and E. M. Belding-Royer, "A study of dynamic addressing techniques in mobile ad hoc networks," *Wireless Communications and Mobile Computing*, vol. 4 (3), pp. 315-329, April 2004.

[12] S. Truchat and A. Pflaum, "Reconfigurable consumer direct logistics systems," Proceedings of 14. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS'05), Kaiserslautern, Germany, February 2005.