Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2007, Article ID 80619, 10 pages
doi:10.1155/2007/80619

Research Article

Profile-Matching Techniques for On-Demand Software Management in Sensor Networks

Falko Dressler, Gerhard Fuchs, Sebastien Truchat, Zheng Yao, Zengyu Lu, and Holger Marquardt

Autonomic Networking Group, Department of Computer Science 7, University of Erlangen, Martensstraße 3, 91058 Erlangen, Germany

Received 30 June 2006; Revised 12 December 2006; Accepted 2 January 2007

Recommended by Marco Conti

The heterogeneity and dynamics in terms of hardware and software configurations are steadily increasing in wireless sensor networks (WSNs). Therefore, software management is becoming one of the most prominent challenges in this domain. This applies especially for on-demand updates for improved redundancy or adaptive task allocation. Methodologies for efficient software management in WSN need to be investigated for operating and maintaining large-scale sensor networks. We developed a profile-based software management scheme that consists of a dynamic profile-matching algorithm to identify current hardware and software configurations, an on-demand code generation module, and mechanisms for dynamic network-centric reprogramming of sensor nodes. We exploit the advantages of robot-based reconfiguration and reprogramming methods for efficient and secure software management. The mobile robot system is employed for decision processes and to store the source code repository. The developed methods are depicted in detail. Additionally, we demonstrate the applicability and advantages based on a scenario that we implemented in our lab.

1. INTRODUCTION

In this paper, we demonstrate the need and feasibility of profile-matching methods for operation and management of wireless sensor networks (WSNs). We developed a scheme for on-demand software reconfiguration and node reprogramming for software updates and task allocation issues. In a lab scenario, we implemented and evaluated mechanisms for (1) profile matching using light-weight RPCs and dynamic addressing as a helper function, (2) dynamic on-demand code generation, and (3) network-centric reprogramming [1, 2].

With the proliferation of WSN and sensor/actuator networks (SANET), new application domains appear that make efficient use of such networks [3–5]. Examples range from habitat monitoring to automation facilities. SANETs are exposed to many challenges that have been identified in this area [4, 6], for example, as energy efficiency, security, and efficient self-organization. An overview of the issues in sensor nodes is provided by Culler et al. [7]. Recently, this list was extended to cover software management in WSNs as well. Due to the heterogeneity of employed hardware platforms and the low resources in terms of processing power, available memory, and networking capacities (sensor nodes are usually able to run a single task only) [8], new approaches for efficient software engineering and software management are needed. An overview of software management techniques in WSNs is given by Han et al. [9]. They depict a model consisting of three fundamental components: the execution environment at the sensor node, the software distribution protocol in the network, and the optimization of transmitted updates. We concentrate on software management techniques for WSNs that are dynamic in terms of availability, mobility, and current application demands. Questions such as how to configure, reconfigure, program, and reprogram networked embedded systems such as sensor nodes are discussed by Handziski et al. [10]. Software management for networked embedded systems such as sensor networks has been intensively studied in the last few years [11, 12]. In this paper, we concentrate on the efficient selection and combination of software modules depending on the capabilities of the available sensor nodes and the global application demands.

In the context of the robot assisted sensor networks (ROSES) 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*[1] for general purpose applications in multirobot systems. Part of this work was an interface between the robot systems and our sensor nodes (see below). This allowed us to study the applicability of the ad hoc sensor network for localization assistance [13]. An example for the latter scenario is assistance for maintenance and deployment of sensor nodes as well as for task and resource allocation [14]. Currently, we are investigating methods for adaptive reconfiguration of sensor nodes using mobile robot systems. The major goal is to reprogram sensor nodes specific to their hardware constraints and according to changes in the environment. In order to address these issues, we apply profile matching mechanisms as formally described in [15]. Basically, profiles are used as fingerprints describing hardware and software installations of networked nodes. Based on adequate profile definitions and profile matching techniques, new samples of software can be generated. In this paper, we discuss the applicability of these mechanisms in the context of sensor networks.

The developed profile-matching techniques for on-demand software management can be used in different applications. We implemented a common interface for our robot control system *robrain*. In this paper, we use a scenario consisting of a WSN assisted by mobile robot systems to depict the method in more detail. These robots are used due to various reasons: the robots can store source code for many applications and dynamically generate binaries for the sensor nodes, single-hop reprogramming is more reliable, and security issues are limited to one-hop neighborhoods.

The rest of the paper is organized as follows. Section 2 outlines the related work concerning network-centric node reprogramming and software management for WSNs. Section 3 depicts the envisioned scenario including our lab setup and the basic concepts of profile-based sensor network reconfiguration. Section 4 provides an in-depth discussion of the developed profile matching techniques for on-demand software management. In Section 5, the developed communication protocol is outlined. Finally, Section 6 concludes the paper and outlines next steps of further work.

2. RELATED WORK

In general, two approaches for software updates in sensor networks have been discussed in the literature: (1) multihop network-centric node reprogramming and (2) robot-assisted software management. Work on the first technique was done mainly based on network-centric reprogramming. For example, the Deluge system [16] was developed for reprogramming Mica2 motes. Deluge propagates software update over the ad hoc network and can switch between several images to run on the sensor nodes. A role assignment system was developed at the ETH Zurich [17] 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 [18] introduces software engineering methods for sensor node programming. Incremental network reprogramming was studied by Jeong and Culler [19]. The primary focus of this work was on the delivery of software images over an ad hoc network.

In general, there are many reasons why mobile robots are useful for sensor networks. This has been outlined for example in the context of SANETs [4]. If available, such robot systems can be used for operation and control of the WSN also as robots are quite powerful systems compared to sensor nodes. Additionally, we consider the use of mobile robots for reconfiguring single sensor nodes and, therefore, larger ad hoc sensor networks reasonable due to several advantages besides their purpose for other operations besides WSN maintenance. 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 reprogramming the sensor nodes. First of all, the flooding—even if efficiently controlled—of code over a multihop sensor network not only consumes a lot of energy, it also disturbs the network in its intended operation, that is, congestion due to code transfer might prevent the transmission of sensor readings. The second reason is security. Achieving 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 help to achieve higher level goals such as task allocation as no centralized server system need to be involved. In this context, the term localized refers to sensor nodes that are direct neighbors in the current robot location. The use of multiple robots can even improve the scheme.

The reconfiguration scheme as presented in this paper allows the dynamic adaptation of WSN to changing application demands and corresponding resources constraints. This strongly influences the lifetime and coverage of sensor networks [20–23].

3. PROFILE-MATCHING-BASED RECONFIGURATION

In this section, we depict the envisioned scenario including our lab setup and the basic concepts of profile-based sensor network reconfiguration.

3.1. Reconfiguration concept

The basic ideas of profile-based reconfiguration are summarized in the following. As previously described, we use mobile robot systems for the decision-making process, code generation, and node reprogramming. A *global goal* is assumed to describe the application requirements. Such a goal could state the need to have specific applications available in all parts of the sensor networks. This example refers to the often discussed *coverage problem* in sensor networks [20, 23].

[1] http://robrain.berlios.de.

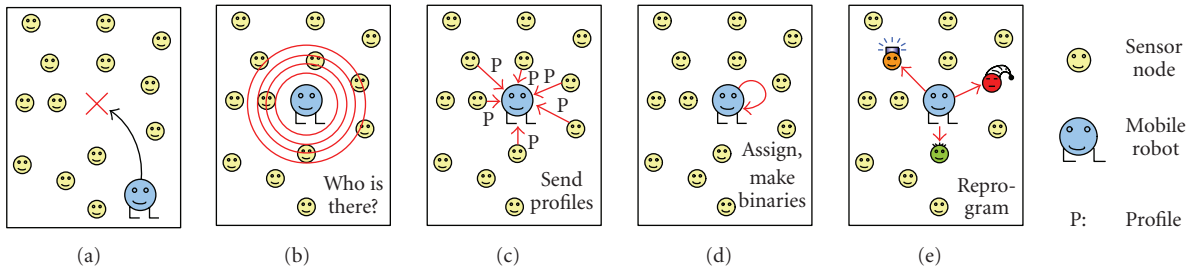

Figure 1: Application scenario for reconfiguration.

Mobile systems are meant to handle this issue [22, 24]. Compared to these approaches, we support the need of multiple applications at once and include the issue of reprogramming. Nevertheless, we do not discuss details of the decision process that is adequately investigated by other groups [20, 25, 26].

Figure 1 shows the principal concept of reconfiguration. We distinguish the following actions.

(a) Depending on the global goal, the robot drives to the position in the sensor network, where reconfiguration might be 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 starting an algorithm for dynamic addressing schemes.
(c) All sensor nodes, 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 nodes, optimized for the current goal. As a result, it creates the new binaries of the sensor nodes. Eventually, additional processing or communication with other entities might be necessary unrelated to the reconfiguration itself.
(e) The robot reprograms selected sensor nodes over the air.

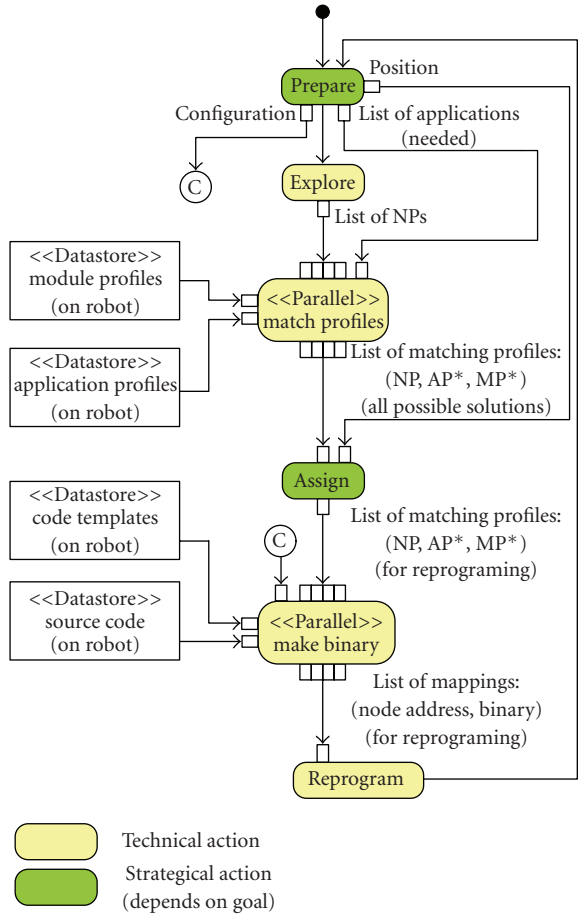

Figure 2: Activity diagram of the reconfiguration process.

3.2. Profile-matching algorithm

The activity diagram of the reconfiguration process of the mobile sensor network is shown in Figure 2. In this section, we use the following notations: NP for node profile, AP for application profile, MP for module profile, and xP^* means at least one profile of type x. Basically, 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.

The reconfiguration process is started with adequate preparation actions. In this step, the robot can perform some initial actions depending on the global goal, for example, moving to a particular position. Then, it determines the current context, that is, requests the profiles of neighboring sensor nodes. Based on this information, the robot works out the configuration and the list of applications that are needed in the current context. The results parameterize the technical actions profile matching and code generation. Finally, all sensors for which new binaries were created are reprogrammed.

Description of the activities

(1) First, `<explore>` is started. As a result, the robot collected all the current configurations of the neighboring sensor nodes in form of a list of NPs.
(2) `<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 nodes are needed. The output is a list of matching profiles. Each entry has the form (NP, AP*, MP*).
(3) `<assign>` reduces the cardinality of the result. This is a strategic action and, therefore, depends on the global goal. The output is the final mapping for the reconfiguration of the sensor nodes.
(4) 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.
(5) Finally, this is taken as the input for `<reprogram>`, which is the last step of the reconfiguration loop. The robot reprograms the sensor nodes over the air.

3.3. Lab scenario

In our lab, we use the Robertino robot platform developed at the Fraunhofer Institute AIS[2] running embedded Linux and Mica2 sensor motes developed at UCB running TinyOS.[3] We have connected an 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. For node reprogramming, we prepared all sensor nodes with an initial binary that contains a module for profile matching concerns. The robot uses this module to query information about the hardware configuration and the currently installed software, for example, 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 while 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. A typical setup of the application scenario in our lab is depicted in Figure 3. Five Mica2 and Mica2dot sensor motes are used to demonstrate the profile matching and reprogramming approach.

4. ON-DEMAND WSN MAINTENANCE: DETAILS AND METHODS

In the following sections, some details on the realization of the profile matching concept for wireless sensor networks are discussed in more detail. We follow the diagram depicted in Figure 2 and outline the single steps.

[2] www.openrobertino.org.
[3] http://www.tinyos.net.

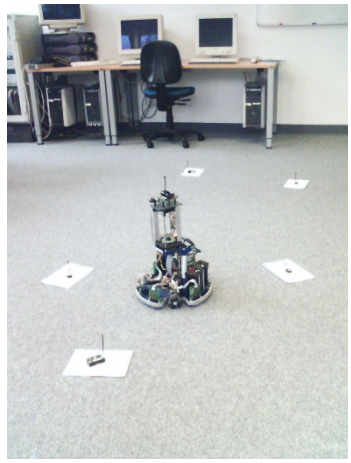

Figure 3: Setup of the lab scenario.

4.1. Preparation

Before the preparation can be initiated, the robot must stop at an adequate position in the area of interest. We do not intend to discuss appropriate mobility models at this place. Such investigations can be found in the literature [22, 27]. The preparation includes issues such as localization of the robot, preconfiguration of system parameters according to the current environmental conditions, and loading application profiles form a local database.

4.2. 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, unique 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. Based on techniques described by Sun and Belding-Royer [28], we developed a localized addressing scheme for this initial task [29]. In the lab, we implemented this addressing scheme using the mobile robot as a local server initiating the round-based addressing scheme.

The second step is to explore the neighborhood. Neighborhood refers to all sensors that are directly accessible without the need for multihop communication. All nearby sensor nodes, that is, all nodes that are alive, in the direct communication range, and available for reprogramming, must be identified and their profiles must be collected. A simple broadcast to the neighboring nodes is used to send a request to all these sensor nodes. Each node sends a reply including its current profile (containing HW/SW info). This profile is taken as input for the following profile-matching algorithm.

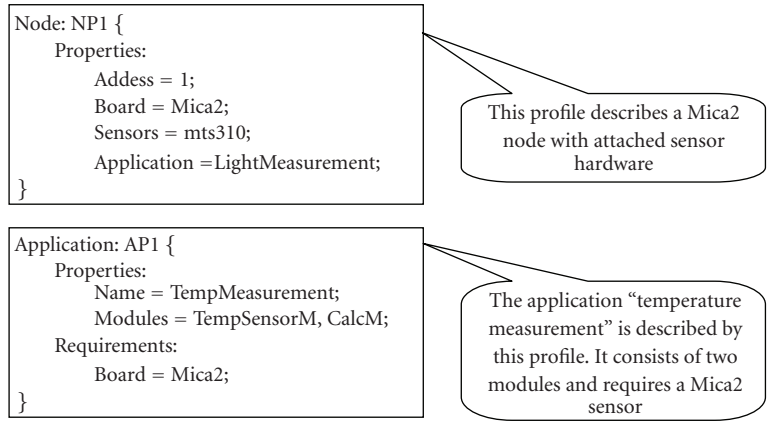

Figure 4: Sample profiles describing HW/SW configuration of sensor nodes and application requirements.

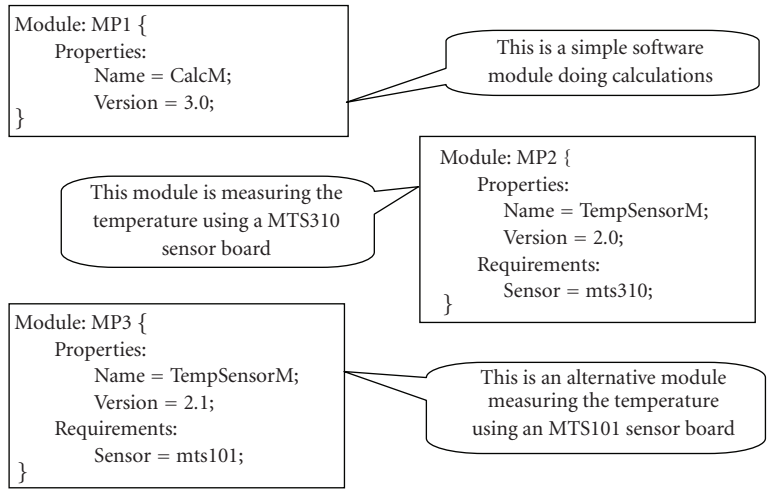

Figure 5: Sample profiles describing single software modules.

4.3. *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 that on the node described by 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 submodules 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 Figures 4 and 5, we present several examples for profiles of nodes, applications, and modules. These profiles are depicted in pseudocode that does not specify implementation details. Representations could be XML, bitmaps, or simple byte arrays (we used byte arrays for our implementation). Please note the importance of having unique names or identifiers for modules and applications. These profiles finally specify the composition of application programs including potential hardware requirements.

In our example (see Figure 4), NP1 is a typical Mica2 sensor mote that has installed additional sensor hardware. The node is used for light measurement. AP1 is an application that measures the temperature. It was developed for Mica2 motes. Similarly, module profiles are used to define the characteristics and prerequisites of single software modules (see Figure 5). MP1 is a hardware-independent module to calculate some statistics of measured data. Finally, MP2 and MP3 represent alternatives of a software module for different hardware systems.

Issues such as versioning and consistency can also be handled by the profile-matching algorithm. Each module description can be annotated with version information and according dependencies (see Figure 5). Questioning the interoperability of different software versions, it can obviously happen that different versions are installed on different nodes. This could lead to interoperability issues. Fortunately, we do not rely on multihop communication for

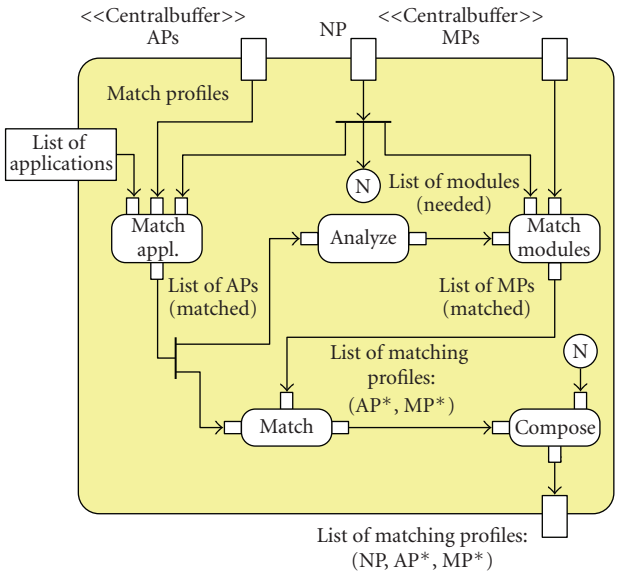

Figure 6: Activity diagram of the profile matching action.

reprogramming (where such issues have catastrophic consequences) but on a very able robot system that can easily provide backward compatibility to older versioned protocols.

An application can be installed if AP matches NP (board property) and all MPs belonging to AP 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 Figure 6.

First, `<match appl>` is initiated. In our example, the operation `[<match appl>: NP1.board == AP1.board → match]` is successful, that is, 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, that is, 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.

4.4. Assignment

A further strategic action is the assignment. Based on the list of matching profiles, the local server, that is, the robot, can use multiple strategies for assigning tasks to specific sensor nodes. We implemented a first match algorithm that might not always lead to optimal solutions. For example, if the battery of a particular node is already in a critical state, communication and computation intensive tasks should not be assigned to this node. Task allocation algorithms can improve this assignment step [30, 31].

4.5. Dynamic code generation

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

Figure 7 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 buffer. Therefore, the code templates are used. `<compile>` compiles all the nesC files. `<compose>` maps the resulting binary with the corresponding node address.

An example is depicted in Figure 8. Needed code fragments, that is, software modules that do not need further adaptation, are compiled to the final sensor application. A special fragment is the base system. Similar to a middleware solution, it provides necessary standard functionality such as the algorithms for profile exchange and network-based node reprogramming. Additionally, code templates can be used representing code that must be adapted according to the local needs, for example, depending on the very specific node hardware.

In our example, code fragments for TinyOS programs are written in nesC. Specific profiles as shown above are connected to these fragments in order to describe functionality and utilization. In order to generate a binary that runs on the nodes described by its node profile, corresponding nesC modules are extracted from the repository and provided to the compiler. 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. Based on the available descriptions, templates can be used for an unambiguous wiring. Secondly, some parts of

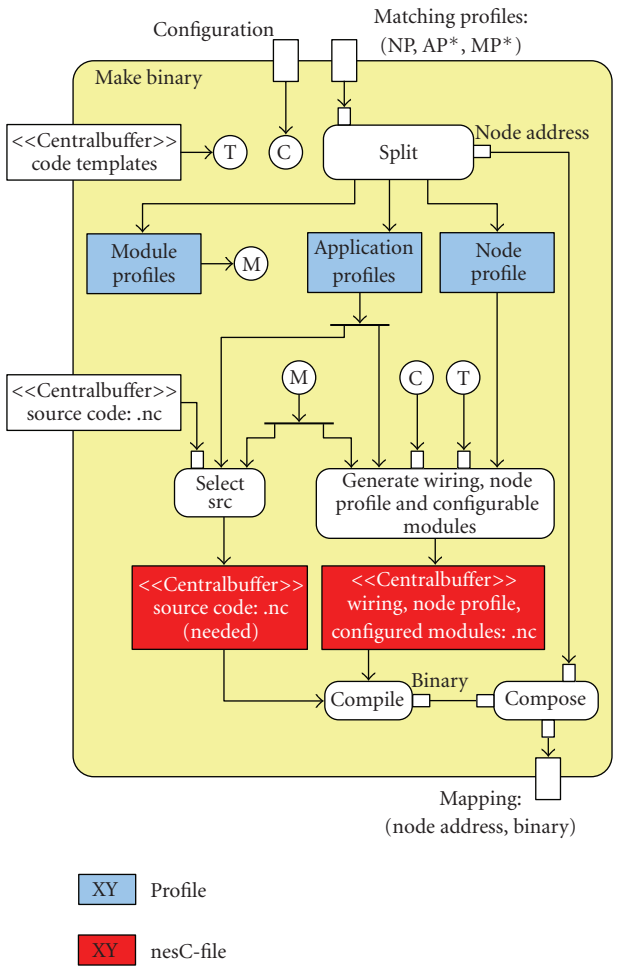

Figure 7: Activity diagram of the make binary action.

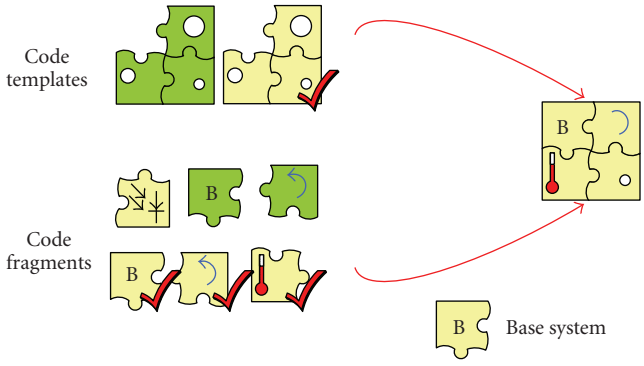

Figure 8: Code selection and complication.

the nesC code have to be adapted to different hardware configurations. We also allow to generate nesC code on demand using code templates. Such templates are filled with variables and algorithms depending on the current context, that is, the environmental conditions. 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.

4.6. Network-based reprogramming

The last part of the node reconfiguration using profile-matching techniques is the reprogramming of the nodes for which new binaries were generated in the last step. We use an extended version of Deluge [32] for this purpose. In the context of code generation, base system software modules will always be installed in any generated binary image. This includes a module that is responsible for network-centric reprogramming and modules for the profile-matching process itself.

5. COMMUNICATION PROTOCOL

Figure 9 depicts the communication protocol used for robot-sensor interactions. It consists of the following protocol messages and states.

(1) *HELLO exchange*

The exchange of HELLO messages is performed in a first step in order to obtain information about the neighboring sensor nodes. This exchange can be initiated by any node. In our scenario, we assume that the robot system will perform this task. Obviously, the HELLO is a broadcast message to be received by all surrounding nodes. The robot maintains an address map with entries for all previously recognized neighboring nodes. Each HELLO exchange is used to update this map.

(2) *Profile exchange*

In order to perform maintenance functions such as software updates, the robot must collect the profiles that describe the current hardware and software configuration of each particular node. A light-weight RPC mechanism is used for this purpose. The local server, that is, the robot system, initiates the call by requesting the current profile from a specific node. Thus, a unique unicast communication is created for each node to be analyzed. The nodes send the requested profiles using a scenario depended encoding. For example, we used an array of 8 bytes to provide the necessary information in the Mica2/TinyOS environment.

(3) *Profile analysis*

After receiving all requested profiles and storing this information in a local map (profile map), the robot can analyze the configuration of all neighboring nodes and decide, whether any update is necessary. Such updates might consist of software updates only. Nevertheless, the more interesting application is to allow performing task allocation schemes by reprogramming adequate sensor nodes.

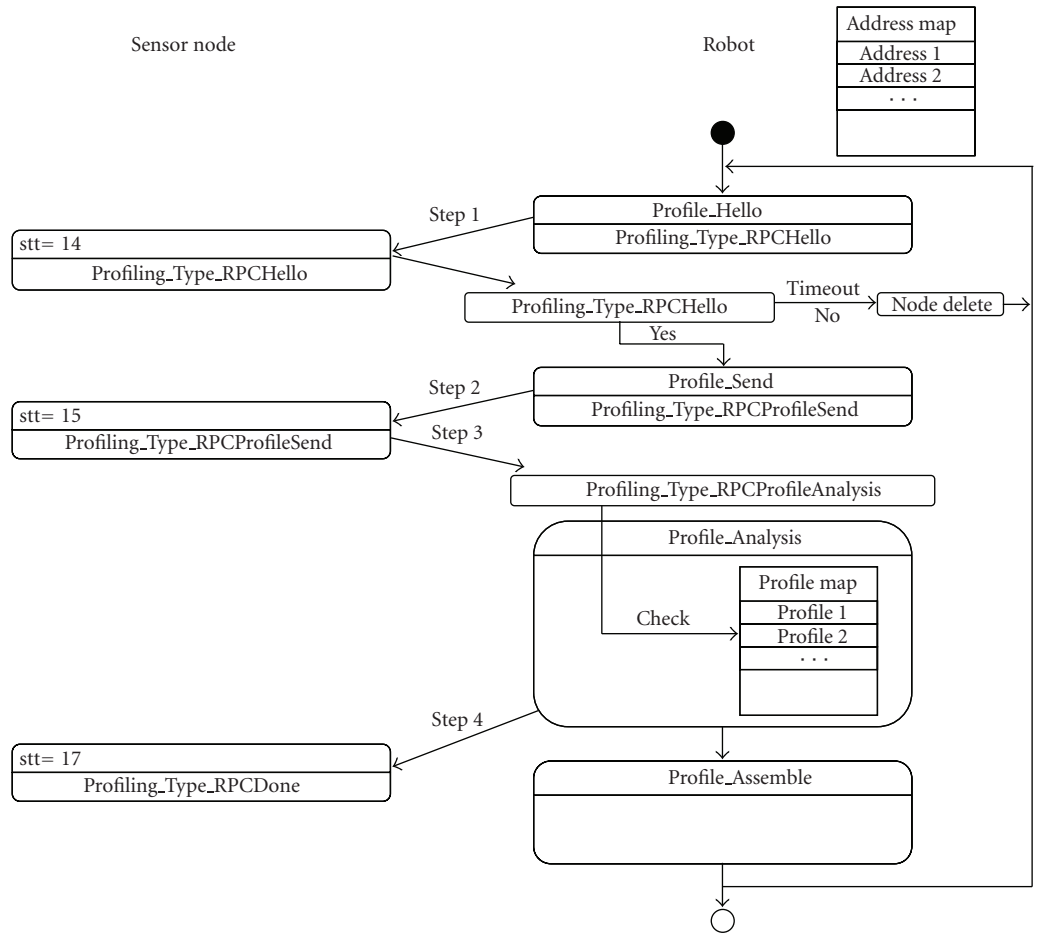

Figure 9: Communication protocol.

(4) *RPC-done/program assemble*

Finally, the RPC is terminated using an RPCDONE message to inform the node that the profile was successfully received by the robot. Depending on the results of the previous profile-matching step, new programs will be assembled and configured by the robot for subsequent node reprogramming.

6. CONCLUSION AND FURTHER WORK

We developed a dynamic software management system that can be applied in different application scenarios in wireless sensor networks. The proposed scheme is based on profile-matching techniques that allow a direct matching of hardware/software capabilities to application requirements. 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 and general-purpose reconfiguration of surrounding sensor nodes.

The presented profile matching 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. Additionally, the complexities of ad hoc communication as well as the security concerns in network-based node reprogramming are minimized. In future and related work, strategies for the robot-based reprogramming must be developed that are optimized for efficiency and coverage.

To demonstrate the functionality of the profile-based reconfiguration scheme, we implemented the following modules: dynamic node addressing, RPC-based profile matching, dynamic code generation, and network-centric reprogramming in our lab environment. Even though we created a common interface for our robot control system *robrain*, the developed systems can be flexibly used. Future work includes the integration of other types of sensor nodes, for example, BT nodes, the development of control schemes for optimized node selection depending on the node state (resources, energy), and the network state (at least in a given region).

REFERENCES

[1] G. Fuchs, S. Truchat, and F. Dressler, "Distributed software management in sensor networks using profiling techniques," in *Proceedings of the 1st International Conference on Communication System Software and Middleware (COMSWARE '06): 1st International Workshop on Software for Sensor Networks (SensorWare '06)*, pp. 1–6, New Delhi, India, January 2006.

[2] Z. Yao, Z. Lu, H. Marquardt, G. Fuchs, S. Truchat, and F. Dressler, "On-demand software management in sensor networks using profiling techniques," in *Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality (REALMAN '06)*, pp. 113–115, Florence, Italy, May 2006.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[4] I. F. Akyildiz and I. H. Kasimoglu, "Wireless sensor and actor networks: research challenges," *Ad Hoc Networks*, vol. 2, no. 4, pp. 351–367, 2004.

[5] D. Culler, D. Estrin, and M. Srivastava, "Overview of sensor networks," *Computer*, vol. 37, no. 8, pp. 41–49, 2004.

[6] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: scalable coordination in sensor networks," in *Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 263–270, Seattle, Wash, USA, August 1999.

[7] D. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo, "A network-centric approach to embedded software for tiny devices," in *Proceedings of the 1st International Workshop on Embedded Software (EMSOFT '01)*, vol. 2211 of *Lecture Notes in Computer Science*, pp. 114–130, Tahoe City, Calif, USA, October 2001.

[8] C. Margi, "A survey on networking, sensor processing and system aspects of sensor networks," Report, University of California, Santa Cruz, Calif, USA, February 2003.

[9] C.-C. Han, R. Kumar, R. Shea, and M. Srivastava, "Sensor network software update management: a survey," *International Journal of Network Management*, vol. 15, no. 4, pp. 283–294, 2005.

[10] V. Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, and D. Culler, "Flexible hardware abstraction for wireless sensor networks," in *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN '05)*, pp. 145–157, Istanbul, Turkey, January-February 2005.

[11] B. Hurler, H.-J. Hof, and M. Zitterbart, "A general architecture for wireless sensor networks: first steps," in *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*, pp. 442–444, Tokyo, Japan, March 2004.

[12] D. L. Martin, A. J. Cheyer, and D. B. Moran, "The open agent architecture: a framework for building distributed software systems," *Applied Artificial Intelligence*, vol. 13, no. 1-2, pp. 91–128, 1999.

[13] F. Dressler, "Sensor-based localization-assistance for mobile nodes," in *Proceedings of 4. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pp. 102–106, Zurich, Switzerland, March 2005.

[14] F. Dressler and G. Fuchs, "Energy-aware operation and task allocation of autonomous robots," in *Proceedings of the 5th International Workshop on Robot Motion and Control (RoMoCo '05)*, pp. 163–168, Dymaczewo, Poland, June 2005.

[15] S. Truchat, G. Fuchs, S. Meyer, and F. Dressler, "An adaptive model for reconfigurable autonomous services using profiling," *International Journal of Pervasive Computing and Communications*, vol. 2, no. 3, pp. S247–S259, 2006, special issue on pervasive management.

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

[17] C. Frank and K. Römer, "Algorithms for generic role assignment in wireless sensor networks," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys '05)*, pp. 230–242, San Diego, Calif, USA, November 2005.

[18] M. Gauger, "Dynamic component exchange in TinyOS (Dynamischer Austausch von Komponenten in TinyOS)," Master's thesis (Diplomarbeit), Distributed Systems, University of Stuttgart, Stuttgart, Germany, April 2005.

[19] J. Jeong and D. Culler, "Incremental network programming for wireless sensors," in *Proceedings of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SAHCN '04)*, pp. 25–33, Santa Clara, Calif, USA, October 2004.

[20] X. Bai, S. Kumar, D. Xua, Z. Yun, and T.-H. Lai, "Deploying wireless sensors to achieve both coverage and connectivity," in *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '06)*, pp. 131–142, Florence, Italy, May 2006.

[21] F. Dressler and I. Dietrich, "Lifetime analysis in heterogeneous sensor networks," in *Proceedings of the 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD '06)*, pp. 606–616, Dubrovnik, Croatia, August-September 2006.

[22] B. Liu, P. Brass, O. Dousse, P. Nain, and D. Towsley, "Mobility improves coverage of sensor networks," in *Proceedings of the the 6th ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc '05)*, pp. 300–308, Urbana-Champaign, Ill, USA, May 2005.

[23] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava, "Coverage problems in wireless ad-hoc sensor networks," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '01)*, vol. 3, pp. 1380–1387, Anchorage, Alaska, USA, April 2001.

[24] M. A. Batalin and G. S. Sukhatme, "Coverage, exploration and deployment by a mobile robot and communication network," in *Proceedings of the International Workshop on Information Processing in Sensor Networks*, pp. 376–391, Palo Alto, Calif, USA, April 2003.

[25] V. P. Mhatre, C. Rosenberg, D. Kofman, R. Mazumdar, and N. Shroff, "A minimum cost heterogeneous sensor network with a lifetime constraint," *IEEE Transactions on Mobile Computing*, vol. 4, no. 1, pp. 4–14, 2005.

[26] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated coverage and connectivity configuration for energy conservation in sensor networks," *ACM Transactions on Sensor Networks*, vol. 1, no. 1, pp. 36–72, 2005.

[27] M. A. Batalin and G. S. Sukhatme, "Sensor coverage using mobile robots and stationary nodes," in *Scalability and Traffic Control in IP Networks II*, vol. 4868 of *Proceedings of SPIE*, pp. 269–276, Boston, Mass, USA, July 2002.

[28] 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, no. 3, pp. 315–329, 2004.

[29] Z. Yao and F. Dressler, "Dynamic address allocation for management and control in wireless sensor networks," in

Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS '07), p. 292b, Waikoloa, Hawaii, USA, January 2007.

[30] K. H. Low, W. K. Leow, and M. H. Ang Jr., "Autonomic mobile sensor network with self-coordinated task allocation and execution," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 36, no. 3, pp. 315–327, 2006.

[31] M. Younis, K. Akkaya, and A. Kunjithapatham, "Optimization of task allocation in a cluster-based sensor network," in *Proceedings of the 8th IEEE International Symposium on Computers and Communication (ISCC '03)*, vol. 1, pp. 329–334, Kemer-Antalya, Turkey, June-July 2003.

[32] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pp. 81–94, Baltimore, Md, USA, November 2004.