TASK MIGRATION SUBSYSTEM FOR
MACRO-PROGRAMMED WIRELESS SENSOR NETWORK

Vitalijus Martusevicius, Egidijus Kazanavicius, Vaidas Jukavicius

Kaunas University of Technology, Department of Computer Engineering,
Studentu st. 50-208, LT-51368, Kaunas, Lithuania, vitalijus.martusevicius@stud.ktu.lt,
ekaza@ifko.ktu.lt, vaidas.jukavicius@stud.ktu.lt

Abstract. Over the past years of research various layers of programming abstraction for networked sensor systems have evolved and the motivation of macro-programming is discussed. Among several recently proposed macro-programming techniques, the abstract task graph macro-programming framework was developed to alleviate application developer work when creating a wide range of wireless sensor network applications. However, its underlying runtime system currently supports only initial task placement onto the nodes of the target network. Since sensor networks operate in a dynamic environment, runtime system should be improved to support the migration of tasks based on energy situation in the networked sensor system. In this paper, an approach for task migration in the runtime is proposed. To achieve this goal, a set of abstract task annotations is extended with migration elements, runtime system is complemented with task migration components, and destination node selection method for the task to be migrated is provided.

Keywords: wireless sensor network, macro-programming, abstract task graph, runtime system, task migration.

1 Introduction

Wireless sensor network (abbrev. WSN) consists of many small independent battery-powered nodes with wireless communication and sensing capabilities. Such sensor networks have a significant potential for various applications which are deployed in diverse areas. Currently a wide range of applications are created for habitat monitoring [23], environmental monitoring [14], target tracking [21], personal health [9] etc.

Since the early days, application developers had to specify application functionality at the level of individual node using programming languages like nesC [8], galsC [6]. In addition, application developer was responsible to ensure desired distributed application functionality. That involves read the values from local sensing interfaces, maintain application level state in the memory, send messages to other nodes and process incoming messages from other nodes. This approach is called node centric programming and is not easy to use in the large sensor networks.

As opposed to the above, sensor network macro-programming provides the ability to specify applications at a high level of abstraction. Abstractions are used to specify the high level collaborative behavior at the system level, while the low level details are controlled by underlying runtime system and are hidden from the developer. Macro-programming languages provide abstractions to specify application behavior which is automatically synthesized into software for each node in the target deployment [3], [10], [17].

Our work is focused on abstract task graph called ATaG [3] and its underlying runtime system [2]. The ATaG is a data driven programming model for application development on networked sensor systems. ATaG is designed to create architecture independent specifications of the application functionality. ATaG runtime system called DART performs software synthesis and enables integration of different protocols and services at the lower layers. Current ATaG compilation framework [18] supports only initial task mapping techniques on the target network. Since sensor networks operate in a dynamic environment, DART should be extended to support task migration based on the rest of energy in the networked sensor system. This involves addressing the questions when to migrate tasks, which tasks to migrate and how to migrate them.

Current ATaG model supports only static logical scopes [16]. In order to specify tasks with migration capability, we extended ATaG notation [3] with dynamic logical scopes. Dynamic logical scopes also enable more flexible application development. Furthermore, we suggested an improvement to the DART with the aim to implement task migration in the runtime. Additionally, a special method is proposed to select a destination node for the task to be migrated. Finally, we evaluated task migration advantages compared with existing initial task mapping technique [19].

2 Related work

Sensor network programming approaches are classified into low and high level models. Low level programming models are focused on abstracting hardware and allowing flexible control of nodes. In this case, the developer has to translate global application behavior into tasks for each node and individually program these nodes. One of the earliest examples in this class is TinyOS [24] which allows application software to access

- 175 -
hardware directly. The logical neighborhood concept is common in node centric programming, because neighborhood creation motivated the design of node level libraries [27], [28]. These libraries handle the low level details of control and provide application programming interface to the developer. Middleware [11], [13] also increased programming abstraction level by providing services to use virtual topologies, logical and dynamic relationships.

High level programming models use an application oriented view and are further divided into group and network level abstractions. Group level abstractions provide a set of programming elements to handle a group of nodes as a single object [26]. In network level abstractions a sensor network is treated as a single abstract machine. Furthermore, network level abstractions are classified into database abstractions and macro-programming languages. The database examples used for high level abstractions are TinyDB [4] and Cougar [5].

Macro-programming languages have greater flexibility compared with database approach and can be used to develop more complex applications. Various functional, data and control driven, imperative and declarative mechanisms are used as a basis for macro-programming languages. Regiment [17] is a declarative functional language with support for region-based functions like aggregation, filtering and mapping. Kairos [10] is an imperative, control driven macro-programming language that provides certain global abstractions and a mechanism to translate a centralized program into a distributed program for sensor nodes. ATaG [3] represents a data driven mechanism as a basis for architecture independent programming of sensor network applications.

A large body of work exists for energy efficient task mapping problem in the networked sensor system. Task placement approaches for unconstrained graphs are presented in [25]. For high level applications a specific greedy solution [1] is suggested. Task placements are specified using roles in [7] article. Authors of [19] present initial task mapping in terms of energy balance and total energy spent.

Finally, MagnetOS [12] system statically partitions a monolithic program into components that are distributed over the sensor network nodes and performs component migration at a runtime. We adapt component migration approach for the DART [2] with the goal to enhance application longevity in the WSN.

3 Dynamic scopes for abstract task graph

The abstract task graph is a macro-programming framework which provides a mixed declarative-imperative approach. As briefly discussed in [3], an ATaG programs are composed of a set of abstract declarations which can be one of three types: abstract task, abstract channel and abstract data. Abstract task and abstract channel have associated annotations. Furthermore, abstract task annotations consist of two parts: instantiation and firing rules. Firing rule specifies when the processing in a task must be triggered, while instantiation rule controls the placement of tasks on the nodes.

Instantiation rules are described in the context of logical scopes. ATaG scoping is thoroughly described in [16] where mainly three additional static constructs are introduced. Authors also notice that supporting dynamic scopes would make a programming model complicated and lead to unpredictable application behavior. We argue with this approach and introduce conditional and task migration logical scopes.

Networked sensor systems monitor an area of interest using nodes equipped with a corresponding sensing device. Time varying properties of the nodes can be evaluated using dynamic scopes. Therefore we complemented particular instantiation rules described in [3] and corresponding scopes [16] with conditional operator using programming language syntax as depicted in expression 1.

\[
\text{nodes-per-instance} : / n [@\text{sensingDevice}] \& (\text{condition})
\]

(1)

Dynamic scopes expressed via conditional operators help to specify application functionality more easily and make a declarative part of ATaG model more flexible. Consequently application developer work with imperative part of ATaG program is reduced.

Some macro-programmed WSN nodes execute more tasks or even more complicated tasks compared with others, i.e. collect and process information gathered from neighboring nodes etc. Thus such nodes go out of battery power quicker than desired and can determine a shortened application lifetime. To increase system and application longevity, we introduced task migration construct \( \rightarrow \) as shown in expression 2.

\[
\text{area-per-instance} : / \text{area} \rightarrow
\]

(2)

Thus, tasks with migration capability are specified by application developer in the ATaG declaration. Migration capability is supported only for tasks, which are assigned to the arbitrarily selected node from a specified area, region or partition. In such cases, initial task mapping problem is solved in [19] article, but authors do not provide a solution for task migration in the runtime.

Sensor network applications usually perform data gathering of some kind. Thus, we created a sample of environmental temperature monitoring application to introduce the syntax and usage of the dynamic scoping constructs. Application functionality we expressed by the ATaG program depicted in figure 1. Normally, data
generated by sensors is collected inside the network before being sent out to the base station. Therefore our ATaG program uses a hierarchical approach for data collection and processing.

Monitor task is instantiated on each WSN node equipped with temperature sensor. This task is responsible for periodic data reading from a temperature sensor every 5 minutes. If sampled temperature value exceeds a predefined threshold, which in our case is 40°C, then temperature data item is added into local data pool for further processing.

To ensure scalability, WSNs are often partitioned into areas. As specified by instantiation rule of the collector task, entire WSN is divided into nonoverlapping areas of 100 square meters. Only one collector task is instantiated in each area, while the exact initial task placement is under responsibility of the compiler. Temperature data items, which are produced on nodes within the same area, are routed to the particular node with the assigned collector task. Furthermore, task migration attribute indicates that collector task can be migrated to either node in the area. Migration is controlled by the underlying runtime subsystem proposed in section 4.

The supervisor task is instantiated on a base station and gathers data from the nodes with the assigned collector task. It analyzes information about temperature violation notifications and triggers required actuators.

4 Task migration subsystem

ATaG runtime system [2] called DART provides the required underlying mechanisms for communication and coordination between instances of abstract tasks. But current DART implementation makes no assumption about network changes in the runtime. Moreover, ATaG programs are synthesized onto WSNs that are dynamic in nature, where nodes fail due to exhaustion of limited energy resources. Therefore, task migration subsystem could significantly improve application longevity by using the available power within the migration area more effectively.

The entire DART functionality is divided into a set of modules to facilitate customization to various deployments. For the sake of completeness each component is described shortly:

- UserTask – represents an application level task which corresponds to abstract task
- DataPool – manages abstract data items in the local data pool
- ATaGManager – stores the ATaG representation that is relevant to the particular node
- Dispatcher – handle the transmission of data items produced on the node
- NetworkArchitecture – maintains topology related information for the node
- NetworkStack – manages network interface of the node

In order to implement task migration in the runtime, we complemented DART with two modules:

- TaskMigration – determines task migration conditions, selects destination node and initiate migration process
- MigrationDataPool – manages all required information associated with task under migration, i.e. saves and restores task related state information during migration process, and controls the dispatch of task environment

Figure 2 depicts intra-node and inter-node flow of control in the task migration process. Inter-node communication corresponds to source and destination nodes of the task under migration. Only DART modules included in the migration process are shown. Individual steps for task migration will be explained.
Every node in the migration area regularly monitors exhaustion of its battery power. In the cases, when a node, hosting the task with migration attribute, detects that its battery level is below the critical threshold, then destination node research procedure is activated. It consists of mainly two functional parts:

- Estimates if the task migration is expedient according to the residual battery power of nodes in the same area. Therefore NetworkArchitecture module is invoked to obtain information about the node’s own location and the composition of its neighborhood in that area. In addition, nodes of the same area are identified by the task instantiation rules that are maintained in the ATaGManager module.
- If task migration is expedient, then destination node is selected using a special method that will be explained in section 5. In general, the destination node is chosen according to the residual battery power and distance from the source node.

Whenever destination node is detected, all required migration information is collected inside the local migration data pool and the runtime system is adapted accordingly. This is a complex process that consists of several actions among components of the runtime system:

- User task code is saved in the migration data pool for transmission to the destination node while own task is disabled and latter removed from the list of tasks running on this node.
- ATaGManager component is reconfigured – information about the declarative part of ATaG program corresponding to the particular task is preserved in the migration data pool and removed from the ATaGManager. This information is composed of input-output channel declarations and annotation of the task to be migrated.
- Data item associated with output channel of the task under migration is also preserved in the migration data pool.
- If source node contains data items that are associated only with the task under migration, then these data items are removed as redundant and their corresponding memory is de-allocated from the data pool.

Figure 2. Control flow in the task migration process
NetworkArchitecture component is reconfigured to cease gathering information about a neighborhood which is determined by channel annotations of the abstract task to be migrated.

Eventually, task migration process is activated and coordinated by synchronization object that controls acquisition and release operations of the data under migration. To handle transmission errors, a specific failure detection technique, similar to the mechanism of forwarding pointers in [12], is to be implemented.

When the migration data arrives to the destination node, it is preserved in the local migration data pool and appropriate instantiation procedure is invoked:

- User task code is restored from the migration data pool, while own task is activated by assigning it to the list of tasks placed onto this node.
- ATaGManager component is updated with the declarative part of ATaG program corresponding to the migrated task from the MigrationDataPool. Thus, ATaGManager is complemented with input-output channel declarations and annotation of the migrated task.
- If the destination node does not contain data items associated with input-output channels of the task, then additional memory is allocated in the data pool for missing data items. Furthermore, if the memory is allocated for a data item associated with the output channel, then its value is restored from the migration data pool.
- NetworkArchitecture component is reconfigured to collect information about the neighborhood, which is determined by channel annotations of the migrated task.

To sum up, we proposed a task migration subsystem, which is intended for the DART. It allows migration of running task from one node to another with minimal disruption to application functionality. The migration operation maintains complete task transactional integrity. Thus, transfers the entire task environment, including task code, associated declarative part, required data items and neighborhood adjustment rules. In the next section we provide a method to select a destination node for the task to be migrated.

5 Destination node selection method

As described in section 3, task with migration capability can be transferred into any node in the migration area. While executing such task in a node, if the remaining energy reaches threshold level, task is migrated into particular node which is healthier. We propose a destination node selection method which is designed to find the most suitable node for the task in respect of remaining energy of the node and task transmission costs.

According to the authors of [15], energy loss in the path \( d_i \) is evaluated as follows:

\[
E(d_i) = p_i \left( \alpha + \beta d_i^\eta \right)
\]

Where \( d_i \) – transmission distance on edge \( e_i \), \( i = 1 \ldots n \), as \( n \) is the number of hops in the route \( R \), while \( R \) is selected by the routing protocol in use; \( p_i \) – total bits transmitted on edge \( e_i \); \( \eta \) – path loss exponent; \( \alpha \) and \( \beta \) – distance independent and dependent energy components for one bit communication. This model is platform independent, because values of \( \alpha \), \( \beta \) and \( \eta \) are to be measured for the WSN platform in use.

Thus, total energy consumption in the route \( R \) we evaluate as follows:

\[
C_R = \sum_{e_i \in R} E(d_i) = \sum_{e_i \in R} p_i \left( \alpha + \beta d_i^\eta \right)
\]

Subsequently, for all the nodes in the migration area, energy balance \( E_B \) is calculated:

\[
E_{B,j} = E_j - C_{R,j}
\]

Where \( E_j \) – remaining energy of the node, \( j = 1 \ldots m \), as \( m \) is the number of nodes in the task migration area. Thus, \( E_{B,j} \) indicates suitability of each node in the migration area for the task to be migrated.

Suitability of every node is evaluated according to the residual battery power and energy costs required to receive the task from the source node.

Finally, the best destination node \( N_{dest} \) for the task under migration is selected as follows:

\[
N_{dest} = \max_j \{ E_{B,j} \}
\]

This method increases application longevity, because particular tasks are adaptive in respect of energy distribution in the WSN.
6 Simulation results

Proposed task migration technique is implemented in a simulation environment and its advantages are explored. Basically, the node that reaches critical level of energy, transfers task with migration capability into healthier node before it dies. To ensure enough time to complete task migration before node goes out of battery power, a migration process is initiated when battery level of the node reaches a predefined critical threshold.

Sensor node parameters in the experiment are selected according to the Sun SPOT node [22] capabilities. It has a 180MHz 32bit ARM920T processor with 512K RAM and 4M Flash memory and is equipped with temperature sensor. Moreover, a linear battery model [20] is used, as it allows evaluate the efficiency of task migration by providing a simple metric of energy consumption for computation and communication.

A sample of environmental temperature monitoring application discussed in section 3 is used in the research. Sensor network is simulated with 24 nodes randomly deployed in the area of 100×200 meters as depicted in figure 3. Maximum transmission range of a node is 50m and the links between nodes represent network connectivity.

According to the ATaG program depicted in figure 1, entire network is divided into two non-overlapping areas of 100 square meters. It is assumed that all nodes are equipped with a temperature sensor and therefore contain monitor task. Furthermore, only two of these nodes have a collector task, which is assigned to the particular node in each area. Initial task assignment is thoroughly discussed in [19] and is out of scope in this article.

To demonstrate task migration process, the migration route of the collector task is illustrated in the first part of figure 3. This route is determined by the algorithm described in section 5. The collector task is initially assigned to node 6. It goes out of battery power quicker than other nodes in the area, because it executes both monitor and collector tasks. Node 6 initiates task migration process when its residual energy reaches 15%. For each node an initial energy of 9000 Joule is assigned.

Using proposed task migration technique, at 1103 minute, sixth node determines that task migration to other node is required before its energy is entirely depleted. The collector task migrates from node 6 through 8, 9, 7 and 4, while available power within the migration area is exhausted.
Remaining energy of the nodes that collector task goes through is shown in figure 4. Because of task migration capability, the collector task operates until 2426 minute and that is 80% longer time compared to the case without task migration feature.

7 Conclusions

In this paper, an approach for task migration in the runtime was presented. At first, ATaG annotations were extended with conditional and migration elements to improve flexibility of the declarative part and to specify tasks under migration. Secondly, the DART was complemented with task migration subsystem. Finally, a corresponding destination node selection method was provided. This technique allows transparently move tasks in the runtime in order to achieve good utilization of available energy in the migration area.

Using results from the simulation, it is verified that proposed task migration technique significantly extends the lifetime of application in the WSN. In addition, application longevity increases with the density of nodes in the migration area.

This work is focused particularly on task migration, but it can be extended to support distributed task execution, when the task under migration is divided into subtasks and executed on several nodes. Thus, even better energy utilization is to be provided.

References


[22] Sun™ Small Programmable Object Technology (Sun SPOT), www.sunspotworld.com.


