Smart Policy Generating Mechanism for Policy Driven Self-Management in Wireless Sensor Networks

Shijie Sun, Bi Zeng, Jianqi Liu

1, 2 Faculty of Computer, Guangdong University of Technology, Guangzhou, 510006, China
3 College of Information Engineering, Guangdong Jidian Polytechnic, Guangzhou, 510515, China
E-mail: srjklssj@gmail.com, zb9215@gdut.edu.cn, iujianqi@ieee.org

Received: 19 May 2013 /Accepted: 19 July 2013 /Published: 31 July 2013

Abstract: Based on the extensive analysis of shortcomings of traditional policy-based wireless sensor network management methods, in this paper, we proposed the SMART507 framework for automatically generate policies which introduce swarm intelligence into the wireless sensor network management field. We discuss the components of the framework in detail which include the policy description language and smart policy generation algorithm. At the end, we test the framework in a wireless sensor network simulator Cooja. The result shows that automatically generate policies can simplify the complexity of network management, enhance the effect of network management and timely response to changes in the network.

Keywords: Policy, Swarm intelligence, Wireless sensor network, Optimization algorithm.

1. Introduction

Internet of Things application created an upsurge in recent years. Due to the perception, data acquisition capability and the low-cost, rapidly deployable, self-organizing, fault-tolerant features, Wireless Sensor Network (WSN) was used in many fields [13-15]. However, each network needs to manage to guarantee its reliable and efficient operation. By adding new middleware abstraction on the WSN node’s software architecture can shield the underlying complexity of the hardware and the network operation details. Also it’s convenient to allow the network administrator to manage WSN through a high-level description of the device within WSN and WSN itself. Starfish [1], FACTS [2], mate [3] are existing WSN management solutions based on middleware. Starfish and FACTS are based on policy, while mate is based on a micro virtual machine. Although these three kinds of scheme can somewhat simplify the WSN management tasks. But they still require a lot management effort to specify the policies or writing management codes. There is a certain distance between the true sense of Autonomic Computing [4] and traditional middleware-based network management method.

In this paper, we combine swarm intelligence algorithm with existing policy-based WSN management solutions to solve a class of optimal management problems in WSN. And we proposed a requirements-based policy automatic generation method SMART507. This approach can enhance the level of autonomy of the Network Management System (NMS) by automatically generated policies according to the management needs.

In this paper, we will describe the SMART507 framework for automatically generate policy which
includes the policy description language; events, actions, conditions' library used in conjunction with the policy description language; and the smart policy generation algorithm. At the end, combined with experimental data, SMART507 will compared with the traditional policy-based network management framework in the following three areas: “extra network traffic”, “node memory requirements” and “ease of use”.

2. Problems in Traditional Policy-based System

Policy-based network management provides a scalable and wide adaptability method for the management of complex systems and sensor networks [5]. A typical policy-based NMS implementation consists of “Event”, “Action” and “Condition” three elements. A simple example is shown in Fig. 1.

```
def (obligpolicy)
on (event)
if (condition)
do (action)
```

(a)

```
oblig on accel_event (acceleration)
do adjust_measurement_interval (5s)
if acceleration <=20
```

(b)

Fig. 1. Policy definition in Finger.

Fig. 1(a) demonstrates the definition of a policy, and Fig. 1(b) shows an example of a specific policy where there is an “accelerated” event triggered, at the same time, if the value of the “acceleration” is not greater than 20, then adjust the sensor measurement interval to 5 seconds. Such a short script with three sentences is able to complete the job usually requires a lot of coding work. That do ease the burden and technical requirements of the system administrator.

Traditional policy-based NMS can run well with universal rule-based network (all nodes in the network have same policies or based on several groups of policies assigned according to roles). But the process of configuration will be very cumbersome for the network which most nodes need personalized configuration. Image the LED lighting control scenario in intelligent buildings. LED types used in the whole building may be up to dozens, while the room types are generally dozens too. Count no personalized lighting configuration requirements, system administrator needs to specify at most “Number of LED types * Number of Room types = Hundreds of” policies. This is undoubtedly a heavy workload. But this is not the worst case. If the customer’s requirements changed, the system needs re-configuration and tuning. System administrators will fall into the tedious work to cope with the endless configuration and tuning.

Based on the analysis of existing problems, we proposed to transform node’s configuration and tuning problem into a combinatorial optimization problem. The only thing that needs network administrator to do is to specify the method of evaluation and the object required to be optimized. Parameters’ setting, policies generation and deployment issues are handled by AI. Therefore, how to cooperate with Smart Policy Generation Algorithm to define an appropriate problem description, how to build the policy generating system’s architecture and how to select policies for each node are keys to the system. We will introduce SMART507 architecture and improved policy description language definition later which upgrade existing policy-based network management system but does not change the node’s implementation to achieve smart policy generation and automatic policy deployment.

3. System Architecture and Basic Concepts

We designed SMART507 architecture and improved the policy specification language in Finger [6]. Fig. 2 shows the overall architecture of SMART507. The system has two parts: Management Server and Node Client.

Management server is equipped with smart policy generation subsystem. It use predefined Actions, Events, Conditions and Influencing Factors follow the problem description provided by user to generate policies and policy deployment method. After the generation process, management server assigns the policies to node clients through the sensor network. Correspondingly, node client is running the policy interpreter. After receipt the policies issued by the management server, the node client update its policy list and execute specified actions when corresponding conditions are met.

Smart policy generation subsystem is the core part of SMART507. To illustrate how it works, we first introduce the concept of Influencing Factors as follows.

**Definition:** Influencing Factors -- Declare the affected parts of the node when the node client performs an Action or an Event happens. Between Actions (or Events) and influencing factors, there are many to many mapping.

According to the characteristics of the policy system, the procedure is nothing more than a certain Event happens, and then the node client check some Conditions, at last it use the given parameters to perform some Actions. For the policies can be generated using combinatorial optimization method, its Conditions and Actions’ relationship are usually fixed, at least the relationship can be specified beforehand based on the pair of “Role / Action”.
And Events can establish relationship with Actions through Influencing Factors. With the above restrictions, the decisions need system administrators to make is to choose what kind of Actions and how to set each parameter to perform the Actions. And luckily, these happens can be done through combinatorial optimization algorithm.

We use particle swarm algorithm [8] in this paper. Let E represents the universal set of elements related to the problem to be solved. E is decomposed into an n-tuple \((E^1, E^2, \ldots, E^n)\). Its union is E, that is \(E = E^1 \cup E^2 \cup \ldots \cup E^n\). Here we will construct all possible pairs of Actions and Nodes to form \(E^1, E^2, \ldots, E^n\) which represent the n-dimensional vector space in candidate solutions. Then run particle swarm algorithm with evaluation function provided by the user. The result is the parameters that each Action needs to set.

So far, the automatic policy generation process only requires the user provide an appropriate problem description. As for the deployment issues, the system get the target node from the pair of "Action / node". Therefore, it simply need to use a certain strategy to assign the policies to the corresponding nodes. Next, we will introduce the expansion Finger syntax first, and give the details of the smart policy generation algorithm after.

### 3.1. Expansion Finger Syntax

The policy’s definition in SMART507 is references to the Finger. Some expansions are made on the Actions, Events and Conditions in the management side to support generate policies automatically.

Before introducing new syntax elements used to describe the policy, we define Problem Description first.

Problem Description consists of five parts (Fig.3):

1) **Goal** (Evaluation Function in fact), used for score the parameters generated by the combinatorial optimization algorithm in each iteration. It provides heuristic information to AI that made intelligent algorithm running toward the direction desired by the user.

2) **Influencing Factors**, used for automatically select the appropriate Actions and Events from the database by AI to constitute a policy.

3) **Chosen Actions** (optional), used for specify what Actions involved in policy generation. If these are not specified, AI made the choice automatically based on influencing factors.

4) **Chosen Events** (optional), used for specify what Events involved in policy generation. If these are not specified, AI made the choice automatically based on influencing factors.

5) **Participating Nodes or Roles** (optional), used for specify the scope of the policy. If these are not description specified, AI made the choice automatically based on the scope of the selected Actions.

---

**Fig. 2. SMART507 system architecture.**

---

**Fig. 3. The composition of the Problem.**

---

Expanded Definition of Action, Event and Condition: Action\((X [ ])\): \(I [ ]\), \(T [ ]\)

-- \(X [ ]\) parameters, the result of the combinatorial optimization

-- \(I [ ]\) influencing factors set, the basis for relevant Actions chosen (Only stored in the management-side database)
-- T [ ] the selectable targets, set of nodes or roles (Only stored in the management-side database)
Event: I [ ]
-- I [ ] influencing factors set, indicates what factors changed or happened to trigger this event (Only stored in the management-side database)
Condition: A [ ]
-- A [ ] default binding Actions to this Condition
PS: [ ] represents a set. Each Action in the Actions library has a predefined Condition set. Of course, the content of this Condition set can be empty. During policy automatic generation process, AI automatically adds corresponding conditions to the Policy.

3.2. Smart Policy Generation Algorithm
Description
The skeleton of the algorithm was shown in Fig. 4, and the details were described below.
1) First, user provides a description of the problem. This description must include “GOAL (Evaluation Function)” and “Influencing Factors”. It may also include optional “Chosen Actions”, “Chosen Events” and “Participating Nodes or Roles”.
2) Check the legality of the user input.
   2.1) Check the legality of the description of the problem. If it is not a valid description, then return an error message to the user, otherwise, the algorithm continues.
   2.2) If the user provide a Chosen Actions set, check whether these Actions contain Influencing Factors include in the problem description. If there was not even one Action contain a relevant Influencing Factor, the algorithm should issues a warning to the user (After all, the evaluation function is user-defined. Contain no relevant Influencing Factors in Actions is not a problem sometimes).
   2.3) Determine whether the participating nodes (or roles) can perform the corresponding Actions. There are two types of mismatches: Some kind of Actions cannot be performed by any node (or role). These Actions are redundant, report the error to the user; some nodes (or roles) cannot perform any Actions, these nodes (or roles) are redundant, report the error to the user.
3) Do semantic analysis on user’s input
   3.1) If the user does not provide the Actions. AI will choose Actions from the Actions library automatically which actions contain only the relevant influencing factors. If there are no relevant Actions in the database, AI provides an error message to the user, then the algorithm terminates.
   3.2) If the user does not provide the participating nodes (or roles). AI will choose participating nodes (or roles) from the Nodes / Roles Library automatically according to the selected Actions in step 3.1. If there are no corresponding nodes (or roles), AI provides the user with an error message, then the algorithm terminates.
   3.3) Chosen the optimization algorithm. If the user specifies an optimization algorithm, the algorithm will configured in accordance with the user's description, otherwise, AI will select and configure the algorithm automatically.
4) Preprocessing of the combinatorial optimization
Here we use particle swarm optimization as a sample to illustrate how to convert Policy generation problem into a combinatorial optimization problem.
   4.1) If the user specifies how to configure the optimization algorithm, it will be configured as the user's description, otherwise, AI will select and configure the algorithm automatically.

Fig. 4. The skeleton of the Smart Policy Generation Algorithm.
5) Combinatorial optimization iteration
   5.1) Perform the optimization iteration once.
   5.2) Use user-supplied evaluation function score
   the iteration’s results (Actions’ parameters). If the
   results meet the threshold condition or the
   number of iterations reaches the boundaries, the
   algorithm jumps to 6.1, otherwise jumps to 5.1.
6) Combine Actions, events and the Conditions to
   form the Policy.
   6.1) Combinatorial optimization result in the
   formation of a series pairs of Action and Node. If
   the user does not specify Events and Conditions
   bound to these Actions, AI will set the policy’s
   type to Execute-Dump that means Event = once,
   Condition = always. Otherwise, set the policy’s
   type in accordance with the user’s description.
   6.2) Integrate the generated policy automatically
   to form a Mission.
7) Verify the generated policy by user before assign it
   to the corresponding node.

4. Simulation

We deploy SMART507 on Contiki [9] micro-operating system, and test it on Cooja [10] WSN simulator with 6LoWPAN network. The simulation network is composed of 30 nodes. Each node has a virtual adjustable brightness LED light. We use SMART507 generate LED control strategy automatically to adjust each node’s LED brightness in the network. The goal is set to under the premise of the effective illumination coverage reached 95 % of the whole area, power consumption is minimal. In order to make the management program to communicate with the nodes in Cooja through an ordinary network interface, we use OpenVPN [11] to establish a tunnel between the boundary router in Cooja and the management program. Fig. 5 shows the LED control effect. Each node is equipped with three LED lights in Cooja. We use red LED (at the top) as the virtual adjustable brightness LED light. Moreover, we created illumination coverage model based on the Cooja’s radio coverage model. Fig. 5 shows the effective illumination coverage and the whole illumination coverage of Node 3.

In the aspect of extra network traffic, we use encoding mechanism introduced in [1]. Each Policy can be loaded into an 802.15.4 frame. For LED light brightness control scenario, each node only need one policy to complete control tasks. And the policy usually does not need to be changed frequently. We only introduce a small amount of extra network traffic during the Policy distribution process. For the example of 30 nodes’ network we took 10 experiments on the network equipped with SMART507 and the network which is not, respectively. Average network traffic is shown in Fig. 6. As can be seen from Fig. 6, the significant differences between two kinds of network’s traffic only exist in network start-up phase and in the 12th minute. That was because the management server distributes new policies to the nodes at these two moments. We can use Trickle mechanism [12] to distribute less urgent policy that reduces the burden on the network further.

In the aspect of node’s memory footprint, because of smart policy generation part only runs on the management server, the memory footprint of the node-side is similar with [1] and [2]. The program size (only include SMART507 part) generated by the mspgcc compiler is: ROM -- 12.23 Kbytes, RAM -- 0.72 Kbytes. Fig. 7 shows the node’s allocation of storage resources. Add extra policies will increase the consumption of resources that need to be analysis according to the specific application. Small footprint shows that deploy policy system on the WSN node is feasible.

In the aspect of ease of use, compared with the traditional policy system, the biggest advantage of SMART507 is that policies does not require manual configuration. AI can automatically generate the appropriate policy on demand. This will undoubtedly simplifies the process of the network’s configuration, improves policy-based network management system’s availability.
5. Conclusions

This paper describes the SMART507 smart policy generation architecture, and tests it in Cooja simulation environment with msp430 microcontroller and 6LowPAN network. The simulation result shows that generate management policies automatically by using the optimization algorithm can simplify the complexity of network management, enhance the effect of network management and timely response to changes in the network and those were all under the premise of small amount of extra network traffic and do not increase the memory footprint of the node. But we should also see this paper only test the policy generation framework under the relatively simple scenario, and in-depth studies needed to be taken in the future.

Acknowledgements

The authors would like to thank the Natural Science Foundation of Guangdong Province, China (No.9151009001000021, S201101001155), the Ministry of Education of Guangdong Province Special Fund Funded Projects through the Cooperative of China (No. 2009B090300 341), the High-level Talent Project for Universities, Guangdong Province, China (No. 431, Yue Cai Jiao 2011), and the Chinese Society of Vocational and Technical Education 2012-2013 scientific research and planning projects (NO. 204921) Guangdong Province Manufacturing, Education & Research Project (2010B090400183) for their support in this research.

References

[7]. W. Cheng, Collection discrete particle swarm optimization and its application in the project resource scheduling, Sun YAT-SEN University, 2012.