Achieving Dependability through Dynamic Reconfiguration in Sensor Operating Systems

Ke Shi*        Xuan Qin        Qifei Cheng        Yidong Cheng
School of Computer Science and Technology
Huazhong University of Science and Technology

Abstract

With the application of sensor networks in industrial, commercial, and medical areas, sensor operating system should be revisited to handle dependability problem. In this paper, we present a novel component-based sensor operating system, MTOS (Micro-Transformers Operating System), which supports adaptive dynamic reconfiguration for resource-constrained sensor nodes to achieve dependability. Through error-driven reconfiguration and change-driven reconfiguration, MTOS can push a system into a stable state in which errors can be recovered with minimized side effects and changes can be safely made without interruption of normal services. The experimental results show that our approach can improve the dependability efficiently.

1. Introduction

Sensor networks have promise for many industrial, commercial, and medical applications. The current and upcoming sensor network deployments for these applications require high dependability infrastructure with the ability to provide expected and quality assured services to multiple users. However, the survey about the early deployed sensor network for academic research could not provide dependable services as the designers expected. Those sensor networks often failed/operated poorly, for example, Great Duck Island network [1] had a median data yield of 58%, and the median data yield of Volcano network is 68% [2]. The following investigations show that one important cause may be the utilized software technology is grossly inadequate to run such complex tasks in uncontrollable environments. Errors in any part of the software can easily bring down an entire network. The resource-constrained feature makes the correctness of software sensitive to the available resources such as memory space, energy. The dynamic environments that impose varying functional and performance requirements can also make the previously properly-functional software fails when the change occurs. In particular, sensor operating system, as a fundamental platform to all the software running on sensor network, should be revisited to handle dependability problem in uncertain environments.

The dependability aspects of sensor operating system and corresponding applications running on it depend heavily on the available resources and the characteristics of the deploying environments, which may change frequently in sensor network’s lifetime. It breaks outside current static approaches that achieve dependability by analyzing models and testing code.

Recently, Dynamic Reconfiguration (DR) has been generating a substantial interest since it allows improving efficiency in the use of system resources, which can impact both on the maximum functionality that the system can execute, on the level of resources needed for a given functionality, or even on the capacity to adapt to changes in the environment. This means that DR may be beneficial to system dependability, especially when a system undergoes variable load situations, when it evolves during its lifetime or even when faults affect part of its structure. Incorporating DR into sensor operating system will allow building highly reliable and resource efficient sensor network applications that are capable of adapting to the environment, to system changes or to different load situations while tolerating the faults, and thus improve the dependability.

It is a challenging task supporting dynamically reconfigurable kernel and applications while considering severe resource constraints of the sensor nodes. Each sensor node has limited batteries, tiny memory pace, and small computing power. For example, Berkeley’s mica motes have only 8 bit processor, 4 KB memory space, and 2 alkaline batteries. Many kinds of operating systems such as TinyOS [3], SOS [4], and Contiki [5] have been proposed for wireless sensor networks. The proposed schemes range from full image replacement to virtual

*Corresponding author, email: keshi@mail.hust.edu.cn
machines. Especially, SOS and Contiki support dynamic loadable module. However, their goal is to reprogram the node, not to achieve the dependability. They are insufficient to reconfigure all software components on the sensor nodes.

In this paper, we present a novel component-based sensor operating system, MTOS (Micro-Transformers Operating System), which supports adaptive dynamic reconfiguration for resource-constrained sensor nodes. In MTOS, user-level applications, kernel features, and device drivers are managed as a form of component, and a dynamic linker is used to link components with each other. Dynamic reconfiguration in MTOS can be divided into two categories: error-driven reconfiguration and change-driven reconfiguration.

Through transparently restarting, error-driven reconfiguration not only makes the failed components recover from the error, but also minimizes the effects on those components dependant on failed components. Change-driven reconfiguration supports replacing individual components at runtime, or to change the compositional topology by adding/removing components and/or changing the patterns of their interconnection, which can push a system into a stable state in which changes can be safely made without interruption of normal services.

The remainder of the paper is organized as follows. Section 2 presents related work. The MTOS and corresponding dynamic reconfiguration mechanisms are described in Section 3. In Section 4, we evaluate our design against the goals identified in the introduction. The experimental results show that MTOS can improve the dependability efficiently. Finally, we offer our conclusions in Section 5.

2. Related work

Component-Based Software Engineering (CBSE) is now recognized for the development of both flexible and well-structured applications, meeting in particular needs for reconfiguration and administration [7]. However, the dynamic reconfiguration and dependability assurance mechanisms only partially fulfill the objective of sensor operating system. For example, the component framework [8] is too heavy for resource-constrained sensor node.

There are some researches about how to improve the dependability of common operating systems [9]. However, the approaches they adopted may cause significant performance overhead that resource-constrained sensor node can not afford.

In sensor network, dynamic reconfiguration becomes an important feature for reorganizing a deployed network or adding new functionalities to several sensors. Considerable research efforts have been made to develop reconfigurable operating systems for wireless sensor nodes. TinyOS [3] only supports full image replacement, which means any error will make the sensor node stop the services in a time period. SOS operating system [4] consists of loadable application modules and a static kernel and adapts position independent code to support loadable module mechanism. Only application modules can be dynamically loaded. Contiki [5] and RETOS [6] have been developed for sensor and embedded systems. They used relocatable code method to relocate and link modules on the sensor nodes. However, all these reconfiguration mechanisms are designed for reprogramming sensor nodes, not for improve dependability.

In [10], Kogekar et al. proposed a constraint-based dynamic reconfiguration scheme in sensor networks. Reconfiguration is triggered by monitoring the system and is performed by transitioning to a new configuration that satisfies current system constraints such as power consumption, latency, accuracy, and other QoS properties. This work is similar to ours; however, we focus on the dependability aspects.

3. Dynamic Reconfiguration in MTOS

3.1. MTOS Architecture

As shown in Fig.1, MTOS consists of core kernel, device drivers, kernel services and applications, which are managed as a form of components.
nodes. The basic data structures include task and component-related structures, free-list of the entire memory space, function and symbol tables. Core kernel enables dynamic component loading and unloading on the sensor nodes at run-time.

Components are position independent binaries that implement a specific task or function. MTOS core kernel facilitates loosely coupled components through passing of memory ownership and efficient function and messaging interfaces. Most development occurs at the component layer, including development of drivers, protocols, and application components. Modification to the MTOS kernel is only required when low layer hardware or resource management capabilities must be changed.

An application in MTOS is composed of one or more interacting components. Many applications are created by combining already written and tested modules. Policy engine is in charge of managing dynamic reconfiguration process, which performs tasks that detect internal and external changes to the system, reflect on the event occurrences, and adapt to the new conditions. Dynamic reconfiguration in MTOS can be divided into two categories: error-driven reconfiguration and change-driven reconfiguration.

### 3.2. Error-driven Reconfiguration

If an error causes the failure of an OS service component, error-driven reconfiguration occurs to make failed component transparently restartable. This is achieved by partitioning component-related OS state into isolated per-component memory regions. Access to these memory regions is provided to OS components on a “need-to-know” basis when processing component requests. Without error-driven reconfiguration, all components dependent on this failed component will be affected. For example, if the network component is terminated because of an error, all running components that depend on the network also fail.

Error-driven reconfiguration decouples an OS service component from application components so that an error in the OS service component does not affect all dependent components. This is achieved by relocating component-specific state from the OS service components into special per-component memory regions referred to as Call State Regions. This is illustrated in Fig.2. In the figure, which represents state information about Component1 maintained by ComponentA, has been moved from ComponentA to Component1’s CSR-A. With this design, error propagation can be minimized by controlling access to the CSR.

When a component calls an OS service component, the kernel grants the OS component temporary access to the CSR. The OS service component can only manipulate data in the CSR while processing the call. When the OS service component completes processing the call, its permissions to the CSR are revoked. If an error occurs during the processing of a call, it is confined to the OS service component’s address space and therefore only affects the CSRs that are currently mapped in. When an error is detected, the OS service component is immediately restarted. The kernel preserves CSRs through a restart. After restarting, the OS component is responsible for detecting corrupted CSRs and repairing them. If corrupted CSRs can not be transparently repaired, error codes are returned for pending calls. The restarted OS components seamlessly resumes servicing those application components whose CSRs were not corrupted. Thus, in our design, only a small subset of the dependent components is affected. We assume that an OS service component restart eliminates the error condition and returns the system to a usable state.

A small increase in the complexity of service component code is unavoidable when using our approach. We feel that this is a reasonable price to pay for increased dependability. Our design also incurs performance overheads due to the need to map memory spaces multiple times within the OS when processing some requests. This approach may also be utilized in the application components that serve the call of other application components, which is decided by what dependability level we want to reach and what overhead we want to pay.

If there are bugs in some components, error-driven reconfiguration also allows problematic components to be updated with newer and more stable versions. And troublesome components can be identified and removed from a node until trouble-free version is provided.
3.3. Change-driven Reconfiguration

When the available resources, deployed environments or performed tasks change, change-driven reconfiguration occurs to make the operating system and the applications running on it adapt to new function or performance requirements by dynamically inserting, updating or removing designated components with minimal interruption of node services. Change-driven reconfiguration can be classified into three phases:

1. Detecting what kind of changes occur to decide what kind of modifications can be done. In particular, decide the designated components that should be removed, updated or inserted to introduce the new functions or improve the performance.

2. Deciding reconfiguration plan to make sure how those designated components removing, updating and inserting are performed. Reconfiguration plan confirms the sequence of those actions based on resource and logic constraints. For example, if a new component N is dependent on another new component M, M should be inserted before N. A dedicated memory space occupied by an old component P is allocated to N, P should be removed before N is inserted.

3. Executing reconfiguration plans to make the system transfer from the old state to a new stable state.

A Policy-based approach is utilized to achieve the above reconfiguring process. A policy engine is to continuously monitor, schedule, and enforce appropriate actions at runtime according to pre-specific policies. Since we can not predict the changes of deploying environments or available resources, a universe set of policies can not be developed in advance. The policies are open to users to specify, analyze, simulate, verify, enforce, and evaluate.

In our prototyping implementations, the policies are expressed as the configuration rules. A reconfiguration rule is a tuple of three elements (e, a, c) where e represents a reconfiguration triggering event, a represents the reconfiguration action, and c represents the configuration description, called Component Configuration Profile (CCP). The CCP includes the fields of type, Components, ID, and reconfiguration indicator of configuration dependency. The policy engine is implemented as a rule-based inferring engine.

The errors can be also considered as a special kind of change, and error-driven reconfiguration can also be considered as a special kind of change-driven reconfiguration and can be managed by policy-based approach. However, error-driven reconfiguration emphasizes the troublesome components can be transparently restartable with minimized effects on the rest components dependant on it.

4. Evaluation

In this section, we present evaluated performance of the proposed MTOS and corresponding reconfiguration approaches. In our experiment, we used Tmote Sky wireless sensor platform.

4.1. Reconfiguration overhead

We used the execution time to denote the overhead of the dynamic reconfiguration. Table 1 shows the subdivided execution time of the reconfiguration schemes when reconfiguring a blink module in Tmote Sky node. The size of a blink module is 512 bytes. We found in terms of execution time our approach performs same with the existing reconfiguration approaches.

<table>
<thead>
<tr>
<th>Costs</th>
<th>Contiki</th>
<th>SOS</th>
<th>MTOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion</td>
<td>924.2 ms</td>
<td>1017.2 ms</td>
<td>1031.1 ms</td>
</tr>
<tr>
<td>Deletion</td>
<td>342.8 ms</td>
<td>378.4 ms</td>
<td>381.1 ms</td>
</tr>
<tr>
<td>link</td>
<td>1.92 ms</td>
<td>1.25 ms</td>
<td>1.82 ms</td>
</tr>
</tbody>
</table>

4.2. Dependability Improvement

A case study is used to demonstrate the effects of our approach to improve dependability. In the case study, 16 Tmote Sky nodes equipped with infrared motion detector consists a sensor network to track the movement of people.

Each sensor generates an electrical pulse based on the difference between the temperature of a heat source and the ambient temperature of the environment. The motion detectors are characterized by their field of view. Based on the sensor measurements, each node computes an estimate of the positions of the people located in its field of view represented as Gaussian distributions. The nodes are placed so that they have overlapping fields of view and they cover the whole space. Neighboring motes communicate with each other the positions that lie on overlapping fields of view and they eliminate duplicate entries based on the proximity between two distributions.

Reconfiguration is driven by short-term/long-term sensing component error. For short-term sensing component error, we assume the node can be recovered from the error by error-driven reconfiguration. For long-term error, the node can not be recovered from the error by error-driven reconfiguration, change-driven reconfiguration is applied. The failed node will inform neighboring nodes. The tracking application in
motes i-1 and i+1 will be reconfigured to account for the loss of the mote. These nodes start computing the velocity for each person. If the position gets out of their field of view, an estimate is maintained by propagating the Gaussian distribution using the latest value for the velocity.

Fig.3 Dependability Improvement and Cost

We use the ratio of successfully tracked people as the indicator of dependability and the energy consumption as the indicator of overhead. As show in Fig.3, the ratio increases from 0.51 to 0.86, almost 50% improvement. At the same time, energy consumption only increases 10%, from 0.91J to 1.01J. Although reconfiguration need extra energy, it make the nodes that long-term errors occurs stop working and communicating with other good nodes, and thus save the energy. The results show our approach can improve the dependability efficiently.

5. Conclusion

We presented an approach to improve the dependability of sensor network applications through dynamic reconfiguration in sensor operating systems. In particular, a novel component-based sensor operating system, MTOS (Micro-Transformers Operating System), which supports adaptive dynamic reconfiguration for resource-constrained sensor nodes is proposed. In MTOS, error-driven reconfiguration not only makes the failed components recovery from the error, but also minimizes the effects on those components dependant on failed components. Change-driven reconfiguration supports replacing individual components at runtime, or to change the compositional topology by adding/removing components and/or changing the patterns of their interconnection, which can push a system into a stable state in which changes can be safely made without interruption of normal services.

However there are additional challenges that need to be addressed. Sensor networks operate in dynamic environments and hence applications must be reconfigured relatively fast. How to design a policy engine with high efficiency is subject to future research. Robustness of the reconfiguration method is also a significant challenge.

6. Acknowledgement

This work is partly supported by the National Natural Science Foundation of China (Grant No. 50805058).

7. Reference