Building Scalable and Robust Architecture for Ubiquitous Sensor Networks with the help of Design Patterns

Syed Obaid Amin, Muhammad Shoaib Siddiqui and Choong Seon Hong
Department of Computer Engineering, Kyung Hee University
obaid@networking.khu.ac.kr , shoaib@networking.khu.ac.kr and cshong@khu.ac.kr

Abstract

Design patterns address a recurring design problem for a specific situation, and present a solution. Design patterns have proven useful in many engineering disciplines such as Software Engineering, Business Information Processing, Architectural Design and so on. However, these patterns have not addressed sensor network specifically. With a growth of sensors and sensor networks, and considering their profound applicability, there is a crucial need to articulate ones experience of application development or deployment of sensor nodes in the form of design patterns to avoid the future mistakes. This paper discusses the same issue and show applicability of design patterns in ubiquitous sensor networks.[1]

1. Introduction

Shared vocabulary is a basic ingredient of any science or engineering discipline. It helps people of the same domain to exchange ideas more conveniently and easily. This vocabulary makes possible to explain the whole scenario with a single word. Design patterns, first introduced by Christopher Alexander [3], are a way to enrich this lexicon. According to Alexander, solutions of the daily life problems have a specific pattern which may be applied repeatedly to solve a problem. In design patterns, we articulate ones experience in a precise set of suggestions in the form of catalog. This catalog suggests that what measures should be taken and what steps should be avoided to complete any specific task. So that others do not make the same mistakes again and again and don't invest their time to rethink on the same problem. As it is said, good judgment comes from good experience, these experience enriched catalogues help novice user to get the deeper insight of their relative domains within the short period. Alexander proposed design patterns for roads, bridges and architectural world but his statement lies true for almost every aspect of life. Therefore, since its advent, design patterns have proven useful in many engineering disciplines such as Software Engineering, Business Information Processing, and Architectural Design etc.

With the inception of sensor networks, a new class of research topics is unleashed. Many of the differences lie almost in every layer of sensor networks as compare to traditional TCP/IP networks. Since sensor networks are in their evolutionary state, most of the research focus on the bottom four layers i.e. Physical, Link, Network and Transport layers to provide a high level of compatibility to the future applications of sensor networks and leave the application layer unattended. As sensor networks are application specific, we cannot find a generalized way of application development in them. However, with a growth of these miniaturized devices and considering their profound applicability, there is a crucial need of articulation of ones experience, of application development or deployment of sensor nodes, in form of catalog to avoid the future mistakes.

This paper elucidates the use of design patterns in sensor networks. The following section outlines the possible classification of design patterns for sensor networks. An issue which is still in veil and haven't got much attention so far.

2. Classifications

2.1. Identification and documentation of existing design patterns:

Creating a design pattern catalog can be divided in to two steps, identification of existing patterns in sensor networks and document them in a design pattern way. However, if this step is failed, the next step is to propose a new pattern and document them eloquently. Both of these issues require a great deal of research and would help novice user to grasp the key design issues more easily and swiftly.

[1] This research was supported by the MIC under the ITRC Project support program supervised by the IITA (IITA-2006-(C1090-0602-0002))

The identification of these patterns could be at any level, within a sensor node, within the object(s) controlling the sensor nodes or at a system as a whole. For example, sensor networks software applications have some different trends than general programming paradigm. Strict energy constraints and limited resources compel developers to write application-specific services. Although, specialized software solutions enable developers to build efficient systems, but on the other hand, they have to compromise the reusability of software components. Features like, late binding or polymorphism may also not available in today's Sensor OS platform. The perfect example of this is TinyOS, in which interaction between components is defined at compile time rather at run time [2], as happens in OOP (Object Oriented Programming) realm. Therefore, it is difficult, although possible, for sensor network programmers to apply OOAD (Object Oriented Analysis and Design) or any other software engineering patterns easily and with an assurance that they will get the same results. These factors increase the need of a new set of design patterns specifically tailored for WSN (Wireless Sensor Networks).

2.2. Application of patterns of other domain:

Design patterns are not a new concept and have been used extensively in many other engineering fields. Although sensor network have some specialized attributes, even then we can apply design patterns of other domains in sensor networks. For example, sensor networks posses some generalized properties of real time embedded systems. Issues which are common in both domains and have been resolved in real time embedded systems along with documentation can be very helpful for sensor networks as well.

The evident use of design patterns is in OOAD (Object Oriented Analysis and Design) where they are used to manage the objects, their relationships and operations. The major work of this domain is by Erich Gamma et al. [1], commonly known as GoF Design patterns. In [1], they capture the experience in designing object-oriented software as design patterns. They named, explained, and evaluated important recurring designs in object-oriented systems and presented a catalog for OOAD.

Being a member of a real world objects, each sensor can also be represented as an Object of OOP (Object Oriented Programming) realm. Each sensor does the same basic things – sensing the environment and responding it, which is analogous to the 'methods' of an object. Based on these actions sensor may alters its state as well which can be regarded as 'attributes' of an OOP object. Design patterns have proven their potential in the OOP domain; and during modeling if we regard each sensor as an OOP object then the application of similar patterns would enhance the management of sensor networks and we could come up with good sensor network design. Of course, not all of the design patterns of OOAD world are applicable in this scenario because some patterns deals with abstraction as well. However, a larger set of OOAD patterns could help us not only in application programming for sensor networks but also in the management and deployment of sensor networks.

3. Design patterns for Sensor Networks:

This paper mainly elaborates the applicability of the design patterns of other domain on sensor networks. Due to page limitation, we would confine ourselves to GoF design patterns and show their applicability in sensor networks at different operational levels. Each section starts with the pattern name, follows with its intent and then outlines its motivation with respect to the sensor networks. We also discuss that when they are applicable in any given scenario. Each section also provides a pictorial description to illustrate the idea.

3.1. Observer Pattern:

Intent:

"Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically" [1]

Motivation:

Consider a weather forecast system, which informs its users about the current weather conditions. Such as, temperature, humidity, barometric pressure and so on. It is clear from the given scenario that we will need some multipurpose sensor or multiple sensors having specific sensing capability to fulfill the requirement. Considering query based approach, in both of the above cases, whenever a client generates any request to take a reading. The query will propagate to sensor node and back. In case of thousands of clients, this situation becomes more cumbersome. To design this architecture more elegantly, we can use "Observer Pattern". Here instead of objects; the one-to-many dependency can be found between the sensor nodes. Each sensor when encounters any change in its surroundings will notify the base station which in turns notifies its registered clients about the weather conditions. This framework can be fine tuned by

adding the functionality of selected updates or pull or push methods.

In sensor networks, it is quite possible that there is no connectivity between nodes for a specific period and the data is kept in storage till we get any connectivity. Any application that wants to send the data will check the link state continuously. In case of multiple applications each application will perform the same task and hence generate more instruction cycle and drain the power more quickly. An application of similar observer pattern can beautify this design as shown in Fig. 1. Here a single thread is taking care of the connectivity, and all other applications are registered to this thread. This thread, acting as a subject, will notify all its dependent applications, its observers, as soon as it gets any connectivity.

Participants:

Subject: Base station in the weather forecast example. In the later example it is a connection monitoring thread.

Observer: Clients in the first example and applications (which require connectivity) in the second example.

Applicable When:

Use the Observer pattern in any of the following situations:

- When there is dependency between multiple entities of network environment.
- When a change to the state of one sensor requires a change on others and we don't know how many sensors need to be changed.

Consequences:

In sensor network deployment where dependency among network entities is one-to-many, the observer pattern seems a simplified solution and in most of the cases acceptable as well. However, notifying only changes is not sufficient in all cases. Without additional protocol to discover *what changed*, observers need to strive hard to infer the changes. What updates should be broadcasted could be another issue of concern, which is totally dependent on the application in context and any negligence in this step could lead to network congestion.

The benefits are some what similar as discussed in [1]. Such as, the sole responsibility of a subject is to notify its observers only. It doesn't care how many interested objects exist. This gives us a freedom to add and remove observers at any time.

Structure:

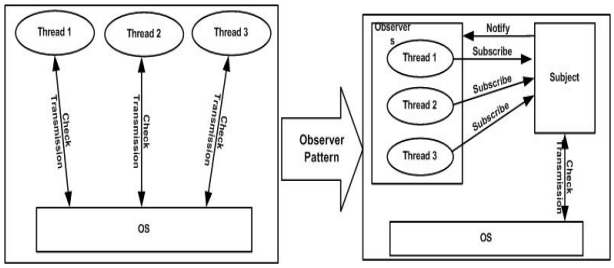

Fig. 1. Example of Observer Pattern

3.2. Proxy:

Intent:

"Provide a surrogate or placeholder for another object to control access to it." [1]

Motivation:

Proxy pattern, also called surrogate, provides a place holder or stand-in for another object. There are a number of reasons why this may be useful in sensor network environment, such as to hide some particular information from the clients and thus any changes would be transparent to the client. On node level, the application of proxy pattern is hard to visualize but have very significant benefits. For example, in some scenarios, dynamically created objects need to be accessed temporarily by other parts of the system, for example image captured by a sensor node. Since some of these objects require large amount of memory; the references to the original object may be shared. Moreover, Virtual Proxy, a variant of proxy pattern, is also useful to controls the access of a resource that is expensive to create.

On the other hand, proxy pattern and many of its variants can be found easily in many existing network designs. For instance in *Protocol Translation,* when proxy can act as an intermediate device that translates data from one protocol to another protocol or in *Controlling Access,* when proxy pattern is used to control the access of a resource based on a user profile or access rights. There are many other uses of proxy pattern, in caching, synchronization, firewall and so on.

Participants:

Proxy: Controls access to the real subject and may be responsible for creating and deleting it.

Client: The entity interested in accessing the resource.

Subject: Defines the real object that the proxy represents

Applicable When:

Use the proxy pattern when in any of the following situations:

- When you want to control the access of desired resource (subject). It could be a sensor node, object with in a sensor node or any other resource.
- When you want to control the access of an object that is expensive to instantiate.

Consequences:

The foremost advantage of proxy in sensor networks is that the clients are simplified. For example, accessing sensor nodes from internet is an active research area nowadays. Use of proxy provided a transparent access to these miniaturized devices. Furthermore, there are usually many fewer client-proxy instances than client-subject instances; the traffic on the communications media is minimized. One message is sent across the bus or network for each proxy, rather than one per client.

Structure:

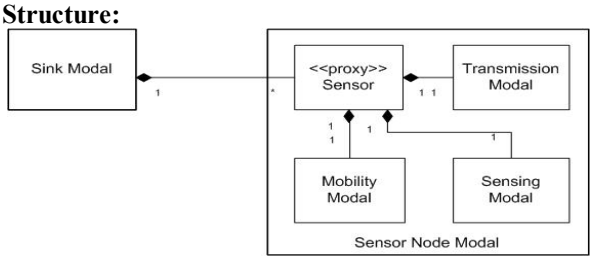

Fig. 2. Class structure of Proxy pattern

3.3. Mediator:

Intent:

- *"Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently."[1]*
- *Design an intercessor to decouple many peers.*

Motivation:

Consider a multi agent system, an agent can be defined as a system which can perceive its environment through sensors and act upon it through effectors [5]. A multi agent system is a collection of these agents. Usually an agent is divided in to different components or modules according to their behavior to enhance reusability. However, such distribution of behavior can result in a structure with many connections between the various components and having too many interconnections tend to reduce the reusability again. Numerous inter-connections make it more difficult for a component to work without the support of others. In addition to that, making significant changes to the overall behavior of the system becomes unnecessarily difficult, since behavior is distributed among many modules. These problems can be avoided by encapsulating collective behavior in a separate mediator module. The mediator controls and co-ordinates the interactions of the various modules within the agent. Each module is only required to know its mediator and all communication with other modules is done indirectly through this channel. The foremost advantages are localization of the overall behavior in one module and the system can easily change its behavior by modifying or replacing just the mediator module.

Another example of mediator can be seen in smart ubiquitous environment where mediator contains the control logic for the entire system. When any new smart appliance is added to a system or any appliance requires a new rule the only place that will require modification would be mediator. Mediator pattern *decouples* all the appliances within the system from each other.

Participants:

Mediator: An intermediary node/component or module which provides an interface for communicating with other nodes/components or modules

Colleagues: Each colleague communicates with its mediator whenever it would have otherwise communicated with another colleague.

Applicable When:

In sensor network scenario is applicable in following situations

- When a set of modules communicate in well-defined but complex ways.
- When it is difficult to reuse a module because it refers to and communicates with many other modules.

Consequences:

Mediator pattern allows system managers to vary and reuse Colleague and Mediator independently. It also simplifies the maintenance of the system and any new functionality can be added at mediator with out affecting colleagues. Moreover, mediator pattern can simplify the communication protocol by replacing many-to-many relationship to one-to-many relationship as it is easy to inspect one-to-many relationship. A drawback of the mediator pattern is that without proper design the mediator itself can become overly complex.

Structure:

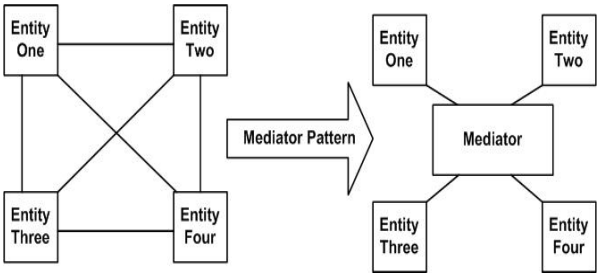

Fig. 3. Mediator pattern

4. Conclusions:

Design pattern have shown their potential in many engineering disciplines. However, their use on sensor networks hasn't been addressed so far. In this paper, we provide pioneering work in this regard. This idea is still in evolutionary stages and needs a great deal of refinement. Due to space limitation, we are only able to provide a gist of an idea and discussed very few GoF design patterns. Many useful and valuable patterns are left to be discussed and will be addressed in future papers. Patterns from real time systems like Recursive Containment Pattern, Hierarchical Control Pattern, and others can also be helpful for sensor networks [4]. This document is a result of our ongoing work and only talks about the applicability of the patterns, articulated for other domain, on sensor networks. However, our future work will not only discuss this issue but will also provide some new patterns specifically tailored for WSN.

5. References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patters: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

[2] David Gay, Philip Levis, and David Culler, "Software Design Patterns for TinyOS", ACM SIGPLAN/SIGBED 2005 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'05), 2005, pp. 40 - 49.

[3] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel., "A Pattern Language", Oxford University Press, New York, 1977.

[4] Bruce Powel Douglass, "Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems", Addison Wesley Professional, 2002.

[5] Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach, Prentice Hall Series in AI", New Jersey, Prentice Hall, 1995.