A Graph Based Approach to Supporting Reconfiguration in Wireless Sensor Networks

Wouter Horré [1], Kevin Lee[2], Danny Hughes[1], Sam Michiels[1], Wouter Joosen[1]

[1] *Distrinet,, Katholieke Universiteit Leuven, 3001 Leuven, Belgium.*
{ danny.hughes, wouter.horre, sam.michiels, wouter.joosen } @ cs.kuleuven.be

[2] *Dieter Schwarz Chair of Business Administration, E-Business and E-Government, University of Mannheim, Mannheim, Germany.*
lee@bwl.uni-mannheim.de

Abstract

Considerable research has been performed in applying run-time reconfigurable component models to wireless sensor networks. The ability to dynamically deploy or update software components has clear advantages in sensor network deployments, which are typically large in scale and expected to operate for long periods in dynamic environments. Realizing distributed reconfiguration in wireless sensor networks is complicated by the inherently asynchronous and unreliable nature of these systems. In such an environment, achieving quiescence is both costly and impossible to guarantee. Additionally, the success of reconfiguration actions cannot be determined with certainty. This paper advocates for a hierarchical, adaptive, graph-based approach to supporting reconfiguration. We argue that application developers should specify only high level reconfiguration graphs, which are then compiled, partitioned and enacted in an adaptive manner by a context aware distributed reconfiguration engine.

1. Introduction

Deploying a large-scale Wireless Sensor Network (WSN) using contemporary technology involves significant expense and development effort. WSNs are thus viewed as long-term infrastructure and are increasingly expected to support multiple applications. Furthermore, applications are increasingly making use of resources distributed across multiple WSNs. Reconfigurable component-based approaches hold significant promise for this application scenario as they allow for dynamic deployment of new functionality along with the modification of existing functionality to meet changing application requirements and environmental conditions.

Consider the following motivating example: a WSN is deployed by the company 'STORAGE_CO'. Initially, this WSN infrastructure supports an application to monitor the location of medical supplies held in a warehouse. After deployment, new regulations are introduced requiring that the temperature of medical supplies be monitored at all times. In a component-based system, this requirement may be met by the *dynamic deployment* of a temperature monitoring component, rather than the wholesale replacement of a monolithic application (shown in Figure 1a). Later, in the same scenario, STORAGE_CO introduces new equipment into their warehouse which generates periodic interference and thus causes errors in location data provided by the WSN. In a component-based system, this problem may be addressed by re-wiring location components via a filter component (shown in Figure 1b).

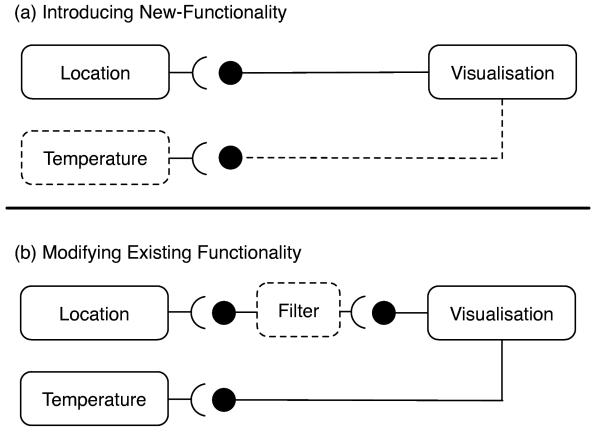

Figure 1: Example Reconfigurations

The simple example discussed above illustrates the benefits of a reconfigurable component based approach in terms of managing changing application requirements and environmental conditions. However, successfully achieving dynamic reconfiguration is complicated by the characteristics of WSNs:

- *Asynchronous and Unreliable Communication:* the low-power network protocols used in WSNs tend to be event-based and unreliable. Thus it is impossible to be certain whether a reconfiguration action has been attempted, or successfully enacted [1].
- *Large Scale*: WSNs may be comprised of thousands of nodes and thus any reconfiguration approach must scale effectively. Partitioning reconfiguration graphs for delegation to agents close to the nodes being reconfigured has the potential to significantly improve scalability.
- *Multiple Owners:* applications may require data from third party WSNs. In these cases reconfigurations cannot be executed directly, but should be partitioned and distributed to the appropriate organization to be enacted based upon their specific reconfiguration policies.
- *Dynamicity*: the limited resources of sensor nodes coupled with the dynamic nature of WSN environments means that no central entity can be assumed to have perfect knowledge of available resources (including those on 3rd party infrastructure).

In this paper we introduce an architecture designed to provide support for the efficient execution of reconfiguration graphs in WSN environments. The proposed architecture models reconfiguration graphs using approaches inspired by *scientific workflows*. Scalability is ensured through *decentralized and hierarchical* execution of the reconfiguration graph and dynamicity is managed through *adaptive* execution of the workflow.

The remainder of this paper is structured as follows: Section 2 provides background on distributed reconfiguration approaches for WSNs and adaptive workflow techniques. Section 3 discusses WSN reconfiguration as an adaptive workflow problem. Section 4 presents the proposed architecture. Section 5 illustrates the appropriateness of this architecture through an example case-study. Finally, section 6 presents some conclusions.

2. Related Work

As previously argued, reconfigurable component models are an excellent tool to support the dynamic deployment, modification and evolution of distributed applications. This can be used to effectively manage changing requirements [4]. This section firstly discusses the state-of-the-art in component models for WSN and then provides background on adaptive workflow techniques.

2.1 Component Models for WSN

Component models for WSN may be categorized as follows:

- *Monolithic*: nodes are re-flashed and re-started, replacing all functionality during the update.
- *Application-based*: units of functionality may be deployed at run-time but support is *not* provided for modifying relationships between functional units.
- *Script-based:* these approaches allow developers to inject lightweight scripts to change the behavior of previously deployed functionality.
- *Component based*: components may be dynamically deployed. Relationships between components may also be modified.

Deluge [5] is a reliable epidemic code dissemination protocol that is used to support *monolithic* flashing of TinyOS [3] motes. Using monolithic re-flashing to achieve only small behavioral changes implies a high energy overhead due to unnecessary data transmission. *Script-based* approaches such as Maté [14] address this by allowing the injection of lightweight scripts which drive the execution of pre-deployed component functionality. The Sun SPOT [6] sensor platforms allow for *application-based* reconfiguration via the deployment of Java ME applications [12]. Contiki [13] provides similar support for the deployment of software modules. While *application-based* approaches offer advantages over monolithic approaches, relationships between applications are opaque and may not be reconfigured. DAViM [15] combine the benefits of an application-based reconfiguration approach with scripting to allow for fine-grained reconfiguration. The *component-based* reconfiguration approach employed in OpenCOM [4] and RUNES [2] provide rich support for reconfiguration, which may involve deploying or updating components as well as modifying component relationships.

We believe that fine-grained component-based reconfiguration approaches hold most promise in WSN environments, though these approaches offer no support for achieving reconfiguration across 3rd party platforms. In addition, these approaches assume that the reconfiguring entity has perfect knowledge of system state, which is impractical. Section 3 discusses how work from the field of scientific workflows may be applied to address these problems.

2.2 Adaptive Workflow Techniques

A workflow represents a group of interdependent tasks, wherein each task may only execute after all the tasks it depends on have successfully completed. When all tasks and their dependencies have completed, the workflow itself is judged to have completed.

Grid-based scientific workflows [7] provide a particularly useful abstraction for the domain of WSN

reconfiguration. The high level goals of a scientific workflow may be described in an abstract form, as a Directed Acyclic Graph (DAG). Such a workflow includes logical entities such as execution locations, logical data (e.g. a logical entity that will later be mapped to a physical location) and logical transformations (e.g. referring to the transformation of logical data to other logical data). A *compilation stage* combines this abstract workflow with specific mappings to locations, files and components.

Describing WSN reconfigurations as a workflow allows us to apply the mature abstractions and conceptual mechanisms of the workflow-processing domain to address the problem of WSN reconfiguration. This is explored in detail in the following section.

3. Modeling WSN Reconfigurations Using Adaptive Workflows

Describing the reconfiguration as a high level workflow and compiling to an executable form rather than directly enacting the process has two critical advantages. Firstly, context-aware run-time optimization can be performed at compile-time. Secondly, abstract workflows allow developers to more easily model complex reconfigurations across heterogeneous platforms.

At the compilation stage, the compiler ensures that the software is deployed in the most efficient way by considering current contextual data. At this point, implicit intermediate tasks may be reified (for example moving components to the location where they should be deployed). At this stage redundant tasks may also be removed from the workflow.

Following compilation of the *reconfiguration graph*, workflow partitioning [9] will be used to split the master reconfiguration graph into smaller *partitions*, which are rendered concrete and distributed to *Action Executors* which have responsibility for achieving reconfigurations in a specific location. Action Executors serve two basic roles. Firstly, they act as units of *virtual synchronicity* [11], arbitrating the success or failure of reconfiguration actions based upon locally gathered contextual data.

During execution of the reconfiguration graph, context-based adaptation occurs at two levels. At the *network level*, the Action Executor, uses contextual data relevant to a specific network is to arbitrate the success or failure of reconfiguration actions. For example, based upon previous performance, the Action Executor may modify the time-out it applies before judging that a reconfiguration message has not been received. Alternatively, the Action Executor may detect and prevent the application of redundant reconfiguration actions.

At the global level, the master workflow executor is informed of reconfiguration progress in terms of the success or failure of concrete action, and based upon the success or failure of reconfiguration actions reported by Action Executors will adapt in a number of ways including: Selecting alternative reconfiguration targets, recompiling the graph using new context data or introducing fault tolerance [10].

4. System Architecture

This section describes an architecture designed to support the adaptive enactment of reconfiguration graphs in WSN environments. Figure 2 illustrates this architecture. It consists, broadly, of a centralized compilation and partitioning stage (Figure 2, top) and a decentralized deployment and adaptation stage (Figure 2, bottom) which is enacted locally on each sensor network.

The reconfiguration process is as follows (the reader should refer to Figure 2). The high-level, abstract reconfiguration graph is compiled to a concrete change graph. A partitioner then splits the concrete graph into concrete partitions. These concrete partitions are transferred to *Action Executers* at the edge of the network, which each have responsibility for enacting changes on one WSN. The Action Executor builds on the approach described in [9], monitoring the state of the WSN using *context-sensors* (lightweight software components that provide status information), which are used to inform the adaptive behavior of the Action Executor, which may include:

- *Target selection*: even when concrete, a target may specify that reconfiguration should be enacted on one of multiple physical nodes. Where this is the case, contextual information such as available battery level may be used to select the most appropriate node.
- *Redundant action removal:* concrete partitions may include unnecessary operations (e.g. deploying a component to a location where an equivalent component is already deployed). The Action Executor will inspect the current network configuration, and where redundant operations are detected, they will not be enacted.
- *Providing Fault tolerance*: based upon previously observed failure rates, an Action Executor may choose to repeatedly apply reconfiguration actions in order to ensure they are successful.
- *Enforcing Reconfiguration Policies*: the owner of each Action Executor will specify a reconfiguration policy that will restrict the reconfigurations that 3rd parties may perform on their WSN infrastructure.

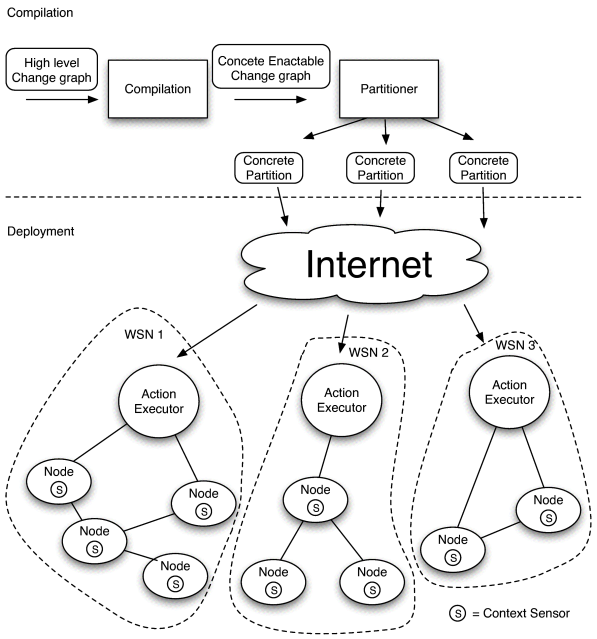

Figure 2: Architecture

5. Case Study Example

This section illustrates the appropriateness of the previously introduced architecture for supporting reconfiguration of WSNs through a detailed case-study. This case-study shows how adaptive processing of the reconfiguration graph can be used to: *i.)* provide fault tolerance, *ii.)* eliminate redundant reconfiguration actions and *iii.)* modify the relationship between existing components. Section 5.1 provides details of the case-study scenario. Section 5.2 shows how adaptive graph processing at the global level can provide fault tolerance. Section 5.3 shows how adaptive graph processing by Action Executors can remove redundant reconfiguration actions. Finally, section 5.4 shows how this architecture can be used to modify the relationships between deployed components. In each case, the XML reconfiguration graph is provided and the reconfiguration process is described in detail.

5.1 Case-Study Scenario

The appropriateness of our reconfiguration architecture will be illustrated through a stock tracking scenario. In this scenario, STORAGE_CO deploys sensor nodes in each of the packages that they are contracted to store. Packages are stored on pallets, which may be inspected at any time by customs officials equipped with mobile devices. Each pallet features one gateway node which runs an action executor and is responsible for all sensor nodes stored in the associated packages. Regulations state that at least half of all pallets stored in the warehouse should report environmental conditions when inspected.

5.2 Providing Fault Tolerance

The first stage in this scenario is for STORAGE_CO to *deploy* a 'PACKAGE_STATE' monitoring component to a subset of pallets in the warehouse (3 and 4). The XML reconfiguration graph for this operation is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<reconfiguration-graph>
  <deploy id="1">
      <component>PACKAGE_STATE</component>
      <location>PALLET_3</location>
      <location>PALLET_4</location>
  </deploy>

  <connect id="2">
      <origin>PALLET_3</origin>
      <dest>BACK_END</dest>
      <component>PACKAGE_STATE</component>
      <component>STORAGE_MONITOR<component
  </connect>
  <connect id="3">
      <origin>PALLET_4</origin>
      <dest>BACK_END</dest>
      <component>PACKAGE_STATE</component>
      <component>STORAGE_MONITOR<component>
  </connect>

  <child ref="2">
       <parent id="1">
  </child>
  <child ref="3">
       <parent id="1">
  </child>
</reconfiguration-graph>
```

Upon execution of the reconfiguration graph, the abstract XML above will be reified to a set of concrete actions. Specifically, the abstract location for 'PALLET_X' will be converted to gateway addresses and the abstract concept *deploy* will be converted to a platform-specific deployment action. The concrete graph will then be partitioned and deployed to appropriate Action Executors. Where an Action Executor reports failure, the global reconfiguration executor will adapt to this contextual information. Thus, the original high level reconfiguration graph will be recompiled, taking into account what actions have been performed successfully and producing a concrete graph that only performs the operations that previously failed. Following successful component deployment the components will then be bound to the back-end package monitoring software of STORAGE_CO.

5.3 Removing Redundant Reconfigurations

During the deployment process outlined above, the Action Executor may also remove redundant actions in the concrete reconfiguration partition. Before each reconfiguration action is enacted, the Action Executor will inspect the specified location and check whether the current state matches the successful outcome of the reconfiguration action to be executed. Where this is the case, the redundant reconfiguration action will simply be omitted. In this specific instance, redundancy may be removed where a matching component (i.e. PACKAGE_STATE) is found to exist, and thus the component will not be re-deployed, conserving valuable resources.

5.4 Modifying Component Relationships

When a customs official with a mobile device arrives to inspect the packages stored by STORAGE_CO, a new reconfiguration graph will be submitted to connect PACKAGE_STATE components within range of the device to the MOBILE_MONITOR component running on the customs official's mobile device.

```
<?xml version="1.0" encoding="UTF-8"?>
<reconfiguration-graph>
  <connect id="1">
      <origin>GLOBAL</origin>
      <dest>MOBILE_USR</dest>
      <component>MOBILE_MONITOR<component>
      <component>PACKAGE_STATE</component>
  </connect>
</reconfiguration-graph>
```

As before, upon execution of the reconfiguration graph, the abstract reconfiguration graph will be reified to a set of concrete actions and dispatched to the appropriate Action Executor to be enacted. New reconfiguration graphs will be dispatched by the custom official's device as it comes within range of new packages / sensor motes in order to deal with mobility.

5.5 Discussion

While the case-study we have presented is simple, we believe that it illustrates the benefits of using a hierarchical, graph-based approach to achieving reconfiguration in WSN environments. In the above scenario, adaptive graph execution was used to provide fault tolerance, while local contextual data was applied to remove redundant reconfiguration operations. We expect that further benefits will be evident in multi-owner WSN scenarios, where our decentralized design will allow each WSN administrator to specify an appropriate reconfiguration policy.

The presented approach has concrete benefits in terms of relieving developers from the complexities of software deployment in unreliable network environments and, moreover, our XML reconfiguration specification language provides a simple, yet powerful mechanism for developers to specify their desired reconfiguration actions.

Another critical advantage of our approach is that it allows for a clean separation of concerns between the *planning* of software deployments and their *realization*. The former should be based upon high level concerns and platform independent, while the latter should be tightly coupled to the specific WSN platform on which deployment is being enacted, such that contextual data can be efficiently exploited.

It is also important to note that, while our case study focused upon a component based reconfiguration approach, our hierarchical, graph-based approach could equally be applied to scripted, application-based or even monolithic reconfiguration approaches.

6. Conclusions and Future Work

This paper has advocated for the use of runtime reconfigurable component models to manage the dynamism of WSN environments. We presented a hierarchical, adaptive, graph-based approach to enacting reconfiguration. This approach draws on existing techniques from the dynamic work-flow processing domain. The appropriateness of this approach was illustrated through a detailed WSN case-study involving diverse reconfiguration actions.

In the short term, our future work will focus upon realizing an implementation of the presented architecture. We will then quantitatively evaluate the potential benefits of an adaptive graph processing to enacting reconfiguration in WSN environments.

In the longer term, we intend to explore the extent to which context-awareness can be used to automatically optimize reconfiguration graphs (for example to exploit currently deployed components).

In summation, we believe that an adaptive, graph-based approach to enacting reconfiguration in WSN holds great potential for lowering the burden on application developers, while allowing for optimization of reconfiguration graphs.

7. Acknowledgments

Wouter Horré is a PhD fellow of the Research Foundation - Flanders (FWO). This research is partially funded by the Interuniversity Attraction Poles

Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven.

8. References

[1] Coulouris G., Dollimore J. and Kindberg T., Distributed Systems: Concepts and Design, Fourth Edition, Addison-Wesley 2005.

[2] Costa P., Coulson G., Gold R., Lad M., Mascolo C., Mottola L., Picco G.P., Sivaharan T., Weerasinghe N., Zachariadis S., The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario, in proc. of the 5th Annual IEEE International Conference on Pervasive Computing and Comunications (PERCOM'07), White Plains, New York, 19-23 March 2007, pp. 69–78.

[3] Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K., System Architecture Directions for Networked Sensors, in ACM SIGPLAN, Vol. 35, No. 11, November 2000, pp. 93-104.

[4] Grace P., Coulson G., Blair G., Porter B., Hughes D., Dynamic Reconfiguration in Sensor Middleware, in the proceedings of the 1st International Workshop on Middleware for Sensor Networks (MidSens'06), Melbourne, Australia, November 2006, pp. 1 – 6.

[5] Hui J. W., Culler D., The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. In proc. of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04), Baltimore, Maryland, USA, November 2004, pp. 81-94.

[6] Sun Microsystems, Small Programmable Object Technology, "Inspiring Java developers to create a whole new breed of devices and technologies - and accelerating the growth of the 'Internet of Things'", available online at: http://www.sunspotworld.com/vision.html

[7] Deelman E., Singh G., Sa M., Blythe J., Gil Y., Kesselman C., Mehta G., Karan V., Berriman G., Good J., Laity A., Jacob J., and Katz D., Pegasus: A framework for mapping complex scientific workflows onto distributed systems, in Scientific Programming, Vol. 13, No. 3, 2005, pp. 219–237.

[8] Gurmeet S., Kesselman C., Deelman E., Optimizing Grid-Based Workflow Execution, in Journal of Grid Computing, Vol. 3, No. 3-4, 2005, pp. 201-219.

[9] Nichols J., Demirkan H., Goul M., Autonomic Workflow Execution in the Grid, in IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Vol. 6, No. 3, May 2006, pp. 353-364.

[10] Lee K., Paton N. W., Sakellariou R., Deelman E., Fernandes A. A. A., Mehta G., Adaptive Workflow Processing and Execution in Pegasus, to appear in Concurrency and Computation: Practice and Experience, 2009.

[11] Schiper A., Birman K., Stephenson P., Lightweight causal and atomic group multicast, ACM Transactions on Computer Systems, Vol. 9, No. 3, 1991, pp. 272-314

[12] Sun Microsystems, Java ME - the Most Ubiquitous Application Platform for Mobile Devices, available online at: http://java.sun.com/javame/index.jsp

[13] Dunkels A., Grönvall B., Voigt T., Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors, in proc. of 29th IEEE International Conference on Local Computer Networks (LCN'04), Tampa, FL, USA, November 2004, pp. 455 – 462.

[14] Levis, P.; Gay, D. & Culler, D., Active Sensor Networks, in proc. of the 2nd USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI'05), Boston, Massachusetts, USA, May 2005, pp. 343 – 356.

[15] Horré W., Michiels S., Joosen W., Verbaeten P., DAVIM: Adaptable Middleware for Sensor Networks, IEEE Distributed Systems Online, 2008, Vol. 9, No. 1