DESIGN OF AN INTELLIGENT MIDDLEWARE FOR FLEXIBLE SENSOR CONFIGURATION IN M2M SYSTEMS

Niels Reijers[1], Kwei-Jay Lin[1,2], Yu-Chung Wang[1], Chi-Sheng Shih[1], Jane Y. Hsu[1]
[1]*Intel-NTU Connected Context Computing Center*
National Taiwan University, Taipei, Taiwan
[2]*Department of Electrical Engineering and Computer Science*
University of California, Irvine, Irvine, CA, USA

Keywords: wireless sensor networks; machine-to-machine; middleware; virtual machine; configuration; policy; profile

Abstract: Most current sensor network applications are built for fixed sensor platforms with a specific wireless network support, building on top of a lightweight OS or even directly on the hardware, and writing applications in an imperative way and from each local node's perspective. This results in software that supports only a specific set of sensors, and difficult to be ported to other platforms. Such software is not acceptable in machine-to-machine (M2M) systems where applications must run on platforms that are inter-operatable and may evolve with time. In this paper we present a project on building intelligent middleware for M2M. The middleware is designed to perform automatic sensor identification, node configuration, application upgrade, and system re-configuration. It allows system developers to specify the application behavior at a higher level, instead of telling each sensor node what to do. A prototype design is presented, as well as the status of our current implementation.

1 INTRODUCTION

Most sensor network projects report that deploying and maintaining a working system is still very hard. (Gluhak, 2011) Many applications are single-purpose systems designed for homogeneous sensor platforms using specific network protocols. The hardware has a fixed set of sensors, and the applications cannot be easily ported to other platforms. The separation of design abstractions between the low-level hardware and high-level applications has not been as successful in sensor-based systems as that for server-based systems, not to mention making them dynamically adaptable and evolvable to new services, and new environments. Except for low level building blocks, there is very little reuse from one project to another.

It is clear that there is a need for better middleware support to ease the deployment of machine-to-machine (M2M) applications (Mottola & Picco, 2011). The goal of our research is to build flexible middleware support so that developers and users of an M2M system do not need to be constrained by which and how sensors have been deployed in a target environment. The built-in intelligence from the middleware can dynamically perform sensor detection, device selection, system configuration, software deployment, and system re-configuration. Like the transition from low-level coding to high-level programming on top of of a general purpose OS and an optimising compiler, we would like to make M2M programming as platform-independent as possible using simple, high-level primitives instead of the node-centric programming.

The main contribution of our project is to support intelligent mapping from a high-level flow based program (FBP) to self-identified, context-specific sensors in a target environment. The automatic mapping capability in the middleware allows an application developer to specify simply **what** types of sensors are needed rather than **which** sensors are used. Such a flexibility is important in order for the same application logic to be adopted by different users at their individual homes or offices with a dynamic, evolving set of heterogeneous sensors.

We present an initial prototype of WuKong, showing its overall architecture, the main middleware components, and basic system size. We have also designed capabilities such as a user policy framework and quality management framework for applications.

2 PROJECT OVERVIEW

To achieve our goals for flexible M2M deployment, we define three orthogonal frameworks:

1. Sensor *profile* framework: to enable the handling of heterogeneous sensor nodes, and for high-level, logical abstraction of sensor capabilities.
2. User *policy* framework: to allow user-friendly specification of application execution objectives, and context-dependent management of system performance.
3. System *progression* framework: to facilitate in-situ software upgrade for dynamically, progressive reconfiguration.

We envision that future M2M systems will have many heterogeneous sensor and actuator nodes, as well as one or more, more powerful gateway nodes for connecting the devices to the outside world. All nodes are connected by wireless technologies such as Zigbee, ZWave and Wifi. One of the gateway nodes will take on the role of configuration and management decision maker, referred as the *Master*, and will be responsible for making deployment decisions and configuring the sensor nodes.

The user runs an M2M application by submitting it to the Master. The Master will then start a discovery phase. Each node is loaded with a device *profile*, identifying its capabilities and characteristics, which the Master can access and configure using the profile framework.

Many M2M applications are heavily influenced by user preferences and system context, since users and objects are often mobile and have changing needs under different situations. For example, a smart home should provide different room temperatures and ambient lighting levels depending on the activity inside of each room. In our envisioned design, users will be able to specify some application *policy* by using a friendly tool or language. The Master will be informed of the user policy and use this information in the configuration decision process for setting up sensors.

Finally, in most M2M applications that are deployed for an extensive lifetime, the Master will need to configure, reconfigure and upgrade software on sensor devices constantly. The management decisions will be made by an optimization engine residing on the Master and may be reviewed periodically or on user request. Reconfiguration decisions will be made based on factors such as sensor profiles, user policy, context knowledge, application history, and real time status, allowing an application to respond to changing conditions.

The part of our middleware running on the nodes, called *NanoKong*, will provide platform independent access at two different levels. First, the profile framework allows the resources, including sensors and software functions, in the network to be discovered by the Master and to communicate with each other through a well defined protocol where the underlying implementation is hidden from the client. Second, NanoKong includes a small Java virtual machine, which will allow us to add application specific behaviour to the functionality already present on the device.

Applications will therefore run in a mix of native code and Java byte code. Obviously the tradeoff here is speed for flexibility. In applications where most of the functionality can be achieved by connecting the resources already present in the network, the application will be running in native code for most of the time, but Java is still used to configure the application. If the application requires processing for which no native support is found, additional Java byte code will be loaded on some devices.

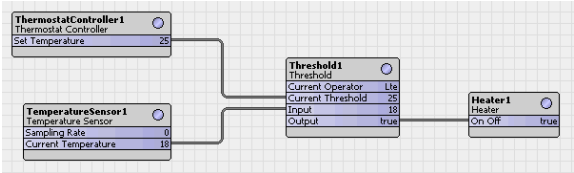

Figure 1: A simple sensor application FBP

3 M2M FLOW BASED PROGRAMMING

Applications in WuKong are developed using a form of flow based programming (FBP). We believe FBP is a suitable model because M2M applications are typically defined by the flow of information between components, as opposed to traditional applications that tend to be more focused on the processing of information. Also, M2M applications are by definition distributed, and the difficulty of managing the communication between devices is one of the main obstacles in M2M development. Using FBP as a programming model allows the application programmer to focus on defining the abstract flow of information in the application, while the resulting FBP program will contain all necessary information to let the framework manage the low level details to physically implement this flow.

The developer constructs an application by selecting logical *components* from a library. Eventually, each component will be realized in practice by a physical sensor or some software running on a device. The decision on how to realize the functionality described by the FBP components

will be made by Master at the time of deployment, but not during FBP construction.

Components in FBP expose their external interface through a set of *properties*. These can be connected using links to create the dataflow in the application. Only properties with matching data types can be connected in FBP.

For example, Figure 1 shows the FBP for a simple scenario to turn on a heater if the temperature drops below a certain value. Besides the temperature sensor and heater, the Thermostat Controller is used by the user to set the desired temperature. This could be a physical remote control or a software program, as long as it conforms to the definition of a Thermostat Controller in our component library. Finally, there is a Threshold, a simple software component, which will output true or false, depending on the desired and measured temperatures, thus turning the heater on or off.

The components an FBP programmer can use to compose the application are defined in a component library in the WuKong framework. Currently, it includes only a basic set of components for commonly-used sensor hardware and functional elements. In addition to these predefined components, a programmer will be able to add custom components containing functions specific to the application if no support is present in the library.

3.1 Links

Since the focus of M2M applications is on the flow of information between components, we believe the links between components should be richer than simple point-to-point connections. WuKong will allow the programmer to specify a variety of options on these links, including:

- *conversions*: to convert one unit into another,
- *filters*: to only propagating "interesting" values (for example exceeding some threshold),
- *reliability constraints*: to indicate whether best-effort delivery is good enough, or a more complex error handling must be employed,
- or inserting *custom code* to do transformations for which no pre-defined support is available.

4 PROFILE FRAMEWORK

While the FBP defines the logical view of the application, the WuKong profile framework tracks the physical resources available in the network and manages the communication between them. The two main concepts in the profile framework are WuClasses and WuObjects. They are related, but not identical, to classes and objects in traditional object oriented programming.

Table 1: WuClass Types.

Usage	Language	Type
Hardware access	C	Hardware
Common processing components	C/Java	Software
Application specific processing	Java	Software

WuObjects are the main units of processing in an application and are hosted on the nodes. The framework has four main responsibilities:

1. Allow the Master to discover which WuClasses and WuObjects are available in the NanoKong on a sensor node;
2. Load WuClasses implemented in Java byte code and create new WuObject instances on a node;
3. Trigger executions within WuObjects, either periodically or as a result of changing inputs;
4. Propagate changes between linked properties of the WuObjects, which may be hosted locally, or on a remote node.

Externally, an WuClass exposes a number of properties describing, and allowing access to, the resource represented by the class. For example the On/Off property of the Heater class is a boolean read-write property. WuClasses are also used to implement various forms of processing. For example the Threshold has three inputs: an operator, a threshold and a current value, and a boolean output indicating if the current value exceeds the threshold.

Besides the properties, WuClasses also export a single update() method which implements the class' behaviour. This function will be called by the profile framework

a) when the value of any property changes, or
b) at fixed intervals, if a sampling rate is set.

Typically, sensors will be triggered at fixed intervals, while actuators like a Heater will be triggered by changes to their input properties.

The properties of an WuObject are managed by the profile framework in a common property store. The framework provides functions for an WuObject to access its data from within the update() method. This allows the framework to monitor the changes an object makes to its properties and propagate them to connected destination WuObjects if necessary.

4.1 WuClass Types

As mentioned before, WuClasses can be implemented either in C (called ***native*** WuClasses), in Java (called ***virtual*** WuClasses), and they can represent either processing or hardware. We can distinguish three main groups (shown in Table 1):

1 Hardware access

WuClasses representing the hardware are implemented in C. Each device will receive a custom built NanoKong which includes the native WuClasses for the hardware present. At startup, an instance is created for each piece of hardware present. For example, a temperature sensor device will have a built-in Temperature Sensor WuObject.

2 Common processing components

In addition to allowing access to hardware resources, WuClasses are also used to implement common operators, such as math and logic operators, the threshold class, etc. In contrast to the hardware WuClasses, software classes will not create any instances at startup, but the Master will use the profile framework to create instances when the application is deployed. These classes be implemented in both C and Java. Many resource rich nodes may be loaded with native versions of commonly used processing classes to allow for more efficient processing. At the same time, there will be a library of Java implementations that can be deployed if no native implementation is available.

3 Application specific processing

Finally, there may be some application specific processing that is not included in the standard library of components. For these cases, the developer can add custom WuClasses using Java.

4.2 Property propagation

The profile framework not only is in charge of running WuObjects and storing their data, but also manages the communication between WuObjects, which may be on different nodes. Since the properties are managed by the profile framework, it can monitor changes and propagate them to connected WuObjects. For example, in a home automation scenario, the Temperature Sensor WuObject is connected to a Threshold WuObject. The profile framework will periodically call the Temperature Sensor's update() method to take a new measurement. After taking the measurement, the update() method will use the functions provided by the framework to update its Current Temperature property. The framework will notice the value has changed, and take care of propagating the new value to the Threshold object's Input property, which may be hosted on another node. This in turn will trigger the Threshold's update() function, causing it to recompute the value of it Output property, etc.

5 COMPILATION AND MAPPING

Figure 2 shows an outline of the application build process. The left part represent the NanoKong VM and the library of virtual WuClasses, both parts of the WuKong infrastructure that will be loaded onto the nodes and Master respectively. The right part shows the compilation process from the application being drawn in the IDE to the Java byte code to be uploaded to the nodes in the network.

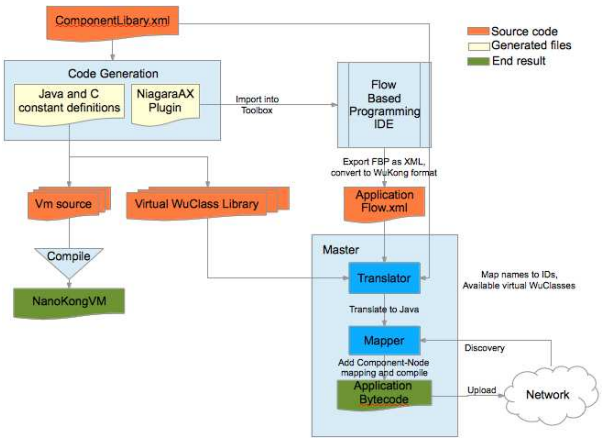

Fig. 2. WuKong application build process

The FBP is exported by the IDE as XML. The Master then generates a Java file, possibly including virtual WuClasses from the library, and wirelessly uploads the compiled byte code to the nodes.

The generated Java code consist of three major parts, two of which are independent of how the application is mapped to physical resources. First, there is a table describing the links between the components in the FBP. Second, some initialisation code is generated for each component to create the instances of software WuClasses and set parameters such as the operator of the Threshold component in our example scenario.

The third part is a table that maps each WuObject to a particular node in the network. To create this table, the Master first starts a discovery phase and asks every node in the network for a list of its WuObject instances and supported WuClasses. The Master will then map each hardware component in the program to an existing WuObject. For each software component, the Master will look for the

Table 2: NanoKong VM code size.

Type	Component	Bytes
VM	Core JVM	5604
VM	Communication (ZWave+Zigbee)	7564
VM	Profile Framework	4244
VM	Native Profiles	632
VM	String operations and IO	4580
VM	Compiled total	20824
Appl.	Home automation application	413
Appl.	Threshold WuClass (C / Java)	199 / 330

corresponding WuClass. If no native class is found, a Java version will be included it in the application.

The mapping can have a large impact on the performance of the application. Developing intelligent mapping protocols will be an important focus in our future work. In our simple home automation scenario, the periodic sampling will create significantly more traffic between the temperature sensor and the threshold components, compared to the link between the threshold and the heater. Thus, the threshold component should be hosted close to the temperature sensor, preferably on the same node.

Also, while in our current implementation there is a one-on-one relationship between the FBP components and the WuClasses, they are two different concepts. Since components in the FBP represent only the logical behaviour of the application, in future versions, a more intelligent implementation may decide to merge several FBP components into a single WuClass or a single FBP component may be translated into several WuClasses if this leads to a more efficient implementation.

6 IMPLEMENTATION

We have implemented a prototype WuKong framework, based on a modified version of NanoVM (Harbaum, 2005), running on the Arduino platform. We have extended NanoVM by adding support for both Zigbee and Z-Wave, adding support for wirelessly uploading a new Java byte code image, and finally of course our profile framework.

Table 2 shows a breakdown of the size of various components in NanoKong. The total size of the VM is currently about 20KB, which allows it to fit onto many of the current sensor platforms.

We compiled the example scenario shown in Figure 2. When a native implementation of the Threshold component is available, meaning no Virtual WuClass had to be included, the resulting size of the Java byte code is 413 bytes.

Finally we examined the sizes of both native and virtual Threshold WuClass. The native implementation is 199 bytes, while the Java version is slightly larger at 330 bytes. This confirms our idea that all but the most resource constrained nodes could be equipped with a basic set of commonly used processing WuClasses to boost performance, and that for classes that are not included in NanoKong, a virtual version can be downloaded.

Figure 3 shows the C implementation of the a boolean ‘not’ WuClass. The wuclass_bool_not struct contains all the information the profile framework needs to manage instances of the class: the global id, a pointer to the update() function and an array defining the properties. In the update() function, we can see how the framework is used to first read the input, and then update the output property. Of the code in Figure 3, the top half is generated from the component definition, and only the update function needs to be hand written. Virtual WuClasses, not shown here, follow a similar pattern in Java.

7 RELATED WORK

There are projects that have addressed some parts of our goals. LooCI (Hughes et al 2012) is the closest to our project by building a reconfigurable component infrastructure. The LooCI component model supports interoperability and dynamic binding. However, it considers component selection to be higher-level services outside of its core

```
uint8_t wuclass_bool_not_properties[] = {WKPF_PROPERTY_TYPE_BOOLEAN+WKPF_PROPERTY_ACCESS_READ  // INPUT
                                         WKPF_PROPERTY_TYPE_BOOLEAN+WKPF_PROPERTY_ACCESS_WRITE // OUTPUT
};
wkpf_wuclass_definition wuclass_bool_not = {WKPF_WUCLASS_BOOL_NOT,         // WuClass id
                                            wuclass_boolean_not_update,    // Update function pointer
                                            2,                             // Number of properties
                                            wuclass_boolean_not_properties // Property datatypes
};
void wuclass_bool_not_update(wkpf_local_wuobject *wuobject) {
  bool input;
  wkpf_read_property_boolean(wuobject, WKPF_PROPERTY_BOOLEAN_NOT_INPUT_VALUE, &input);
  wkpf_write_property_boolean(wuobject, WKPF_PROPERTY_BOOLEAN_NOT_OUTPUT_VALUE, !input);
}
```

Figure 3: Native implementation of an WuClass for a boolean “not” operator.

functionality. ADAE (Chang and Bonnet, 2010) configures, and at runtime dynamically reconfigures, the network according to a policy describing the desired data quality, including fallback options which the system may use in case the ideal situation cannot be achieved. However, a skilled engineer is needed to translate a user's requirements into the constraint optimisation problem for ADAE.

There have been several projects which have developed virtual machines for very resource constrained devices, either using general purpose languages like Python or Java (Brouwers, Langendoen & Corke, 2009, Aslam, et al., 2008, Harbaum), or specialised for WSN (Levis & Culler, 2002, Müller, Alonso and Kossmann, 2007). These technologies allow the application to be developed in a higher level language on heterogeneous networks. However, applications are still written in an imperative way, and from the node's perspective.

Very different approaches are taken by Agilla (Fok, Roman & Lu, 2005) and MagnetOS (Liu, et al., 2005). In Agilla programs consist of software agents that can move around autonomously in the network. While this allows some behaviours to be expressed in a natural way, the paradigm is very different from conventional languages, and the assembly-like instruction set makes it hard to use. Cornell's MagnetOS is interesting because it proposes a novel programming model which allows the user to write the application as a single Java application, which is then automatically partitioned and deployed to minimise energy consumption. However, it requires significantly more computing power on nodes.

8 CONCLUSIONS

In this paper we have presented the design of an intelligent middleware, WuKong, for M2M-based systems. The WuKong middleware provides platform-independent access to heterogeneous resources at two different levels. First, the profile framework allows the resources in the network to be discovered by Master and to communicate with each other through a well defined link protocol. In addition, the middleware includes a small JVM, which will allow developers to dynamically add application specific behaviour to the functionality already present in the sensor hardware.

A working prototype of the WuKong middleware has been developed, including the profile framework. The prototype is able to compile an application, described as a high level flow based programm, map the logical components to physical nodes, and wirelessly deploy and run the application.

ACKNOWLEDGEMENTS

This research was partially supported by National Science Council of Taiwan, National Taiwan University and Intel Corporation under Grants NSC 100-2911-I-002-001, and 101R70501.

REFERENCES

Chang, M., Bonnet, P., 2010. Meeting Ecologists' Requirements with Adaptive Data Acquisition. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*

Levis, P., Culler, D., 2002. Maté a tiny virtual machine for sensor networks. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*

Müller, R., Alonso, G., Kossmann, D., 2007. A virtual machine for sensor networks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*

Brouwers, N., Langendoen, K., Corke, P., 2009. Darjeeling, a feature-rich VM for the resource poor. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*

Aslam, F., Schindelhauer, C., Ernst, G., Spyra, D., Meyer, J., Zalloom, M., 2008. Introducing TakaTuka: a Java virtual machine for motes. In *Proc. of the 6th ACM Conference on Embedded Networked Sensor Systems*

Harbaum, T., 2005. NanoVM http://www.harbaum.org/till/nanovm/index.shtml

Hughes, D.; et al;, "LooCI: The Loosely-coupled Component Infrastructure," *11th IEEE Symposium on Network Computing and Applications (NCA)*, pp.236-243, Aug. 2012

Fok, C.-L.., Roman, G.-C., Chenyang, L., 2005. Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*

Liu, H., Roeder, T., Walsh, K., Barr, R., Sirer, E. G., 2005. Design and implementation of a single system image operating system for ad hoc networks. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*

Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., Razafindralambo, T., 2011. A Survey on Facilities for Experimental Internet of Things Research. In *IEEE Communications Magazine, Vol. 49, no. 11, Nov. 2011*

Mottola, L., Picco, G. P., 2011. Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art. In *ACM Computing Surveys, volume 43, issue 3, April 2011*