2016 IEEE First International Conference on Internet-of-Things Design and Implementation

Extending T-Res with mobility for context-aware IoT

Shashank Gaur, Raghuraman Rangarajan, Eduardo Tovar
CISTER/INESC TEC, ISEP, Polytechnic Institute of Porto, Porto, Portugal
email:{sgaur, raghu, emt}@isep.ipp.pt

***Abstract*—In this paper, we develop a framework for building context-aware applications in IoT. The IoT paradigm brings in various new issues such as macroprogramming, interoperability for heterogeneous devices and in-network processing. Solutions to these issues can enable IoT to support all available resources in an efficient manner and also enable ease of access for users. In addition, this can help in collecting useful information about the user and the system, such as context. Understanding context of different entities and taking actions accordingly will enable a context-aware IoT. However, no complete solution is available to this issue of achieving context-awareness in IoT. In this paper, as a step towards a context-aware framework, we present a mobility-enabling extension of the T-Res programming abstraction. We implement a web-based framework for users to write context-aware applications. We then describe and implement an automated mechanism for deploying these applications.**

I. Introduction

Till now, applications for Wireless Senor Networks(WSNs) were typically designed to perform a single task repetitively, often in strictly controlled environments. This meant that programming applications for a specific system were a one time task. Nowadays, WSNs have become an integral part of the Internet of Things (IoT) paradigm. This enables IoT to become more pervasive, powerful, reliable, and able to incorporate an increasing number of functionalities. Such advancement enables the development of a self-adapting IoT, that can respond to environmental conditions (or context), such as location, activity, availability of resources or time, without need of actions from user. To enable this, there is a need for a framework, which enables the user to think about IoT in an abstract manner and offloads any responsibilities regarding the resources. In this paper we discuss ongoing efforts to provide solutions to this problem. Our contribution is an extension to one of the existing tools, T-Res [1].

This paper is organized as follows. Section II discusses context-awareness in IoT. Section III examines related work for context-aware IoT. Section IV presents our idea of Context Aware Framework (CAF). Section V describes our effort and implementation of the extension of T-Res [1] towards CAF. Section VI provides a simulation based demonstration and finally, Section VII provides concluding remarks.

II. Context-Awareness in IoT

One important challenge for IoT is the ability to exploit the dynamic nature of the physical environment. This can be achieved by deploying a new class of applications that are aware of the context for the users and/or the system itself. Such context-aware IoT can adapt according to the location of the user, the collection of nearby entities, the host infrastructure, or other accessible resources, as well as to the changes in these over time. A system with these capabilities can examine the environment and react to such changes. Context can include energy and noise levels, network connectivity, communication costs, and even the social situation; e.g., whether someone is with their family, friends or co-workers. Three important aspects of context for both user and system are: (1) where you are; (2) when you are; and (3) how you are. These aspects contribute to define a particular context, and that context contributes to define a selection of applications and resources.

The importance of context-awareness has existed before for ubiquitous and pervasive computing [2], [3]. Most of the existing work has been about collecting the data from a limited number of input devices and analyzing it afterwards. With almost every heterogeneous devices equipped with sensors, now context can be understood more clearly than ever before.

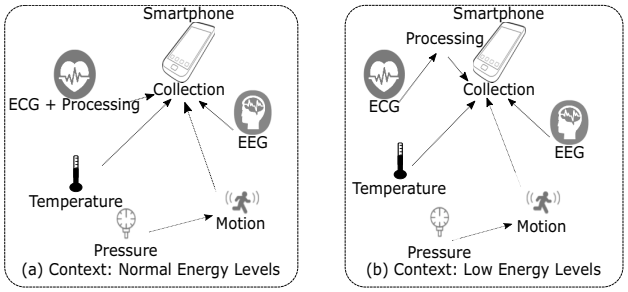

Fig. 1: Context-awareness in body area network.

To understand the importance of context-awareness in IoT, let us consider an e-health scenario, where vital parameters of a patient are monitored by a body area network (BAN). As shown in Figure 1(a), the BAN has various sensor nodes measuring specific vital parameters (Pressure, Motion, Temperature, ECG, and EEG). The sensed data needs to be pre-processed before results are delivered to a medical expert for evaluation. For example in the case of ECG, there is a need to perform some preprocessing such as noise rejections, in order to have valuable information for the medical expert. Let us assume this is done via writing an application with multiple modules. For example the *ECG module* collects the raw data of heart's electrical activity from the sensor and the *Processing module* operates on that raw data to generate valuable information. When the BAN has normal energy levels, all the modules may be deployed on each sensor and the results may be collected by the *collection* on the smartphone for delivery, as shown in Figure 1(a).

However, with time, due to running heavy processing, the energy levels of the ECG node may drop below the desirable levels. To save energy until the node is recharged, the *Processing module* can be moved from the node to the smartphone (Figure 1(b)) and then just poll the sensor for the raw data. After that, the smartphone can process the data into valuable information within the required time. Once again when the node has sufficient energy levels, the module can be moved back to the BAN from smartphone. For such abilities, Abstraction is required for the user to program applications independently of the resources on which the application is executed. Modularity is required in this case to easily break down the application running on the ECG node into pieces,

978-1-4673-9948-7/16 $31.00 © 2016 IEEE
DOI 10.1109/IoTDI.2015.16

while with the help of Mobility the modular piece can be moved to the smartphone when required.

This need can occur in various situations. Nowadays smartphones have capabilities to measure some vital signs. In case of low energy levels of a particular node, the smartphone can take over the complete module from that node until it recovers. Here we are using our e-health usecase just to introduce the intuition of context-awareness, but the basics for context switch can exist in any other foreseeable scenario. Also some relevant research work for BAN application already exists [4].

III. Related Work

Providing a high level abstraction for programming wireless sensor networks has been a major research direction for almost a decade. To support the context-awareness in IoT, there is a need for more efforts in the same direction as some of relevant work, which already contributed significantly towards macroprogramming. To name a few of them, we refer to Regiment [5] and Abstract Task Graph [6].

Regiment is a functional reactive programming model, which treats the outputs of sensor nodes as Streams [5]. A programmer can write functionalities based on these streams instead of worrying about the nodes. There are some basic functions such as *rmap*, *rfilter*, and *rfold* to operate on these streams. The streams can be combined into groups which are called regions. Due to a functional approach, the regiment provides a high level of accuracy in performance.

While Regiment is a Functional approach, Abstract Task Graph (ATaG) is a data driven approach. In ATaG, every application is divided into three declarations: *Abstract Tasks*, *Abstract Data* and *Abstract Channels*. Abstract Tasks represent the type of processing in any application, Abstract Data represents the type of data handled by the applications and Abstract Channel associates the task declaration with data declaration. Using these declarations any application can be described by a model and then that model can be instantiated any number of times throughout the sensor network. ATaG provides abstraction because the number and the placement of the application can be determined at compile or run time according to the target devices.

The above mentioned efforts do not take into account the diversity of any IoT devices included in possible scenarios. The IoT devices bring in a number of new challenges such as in network processing, heterogeneity of software and hardware platforms, resource management, etc. There are other recent contributions that try to contribute towards solving these challenges. T-Res [1] is a programming abstraction targeted specially for IoT. It assigns Uniform Resource Identifier (URI) addresses to devices and utilizes Constrained Application Protocol (CoAP) [7] instructions to deploy applications. We will take a detailed look at T-Res in section V.

These previous efforts still lack the support required for context-aware IoT, which our proposed framework attempts to provide. The programming model described in this paper provides essential features such as Modularity, Mobility and Abstraction, in a way that is absent in any previous work. In this paper, we try to extend T-Res towards CAF by enabling one of those features, i.e. Mobility.

IV. Context-Aware Frameworks (CAF)

As discussed before, IoT can have various contexts constrained by security, power management, communication cost, or user behavior. For any IoT, these contexts can change with time or multiple contexts can exist at the same time. To adapt to new context scenarios the system must take actions. These actions can be the deployment of new applications across the network or re-configuration of old ones. We define a framework with such capabilities as Context-Aware Framework (CAF). A CAF should be able to learn about context by design or by monitoring the system.

To achieve these goals CAF needs to have three features (Figure 2). A programming approach to write context-aware applications, a resource manager to deploy those applications and keep track of resources, and a context manager to learn from current status of applications and resources. In this work we discuss the programming approach required for a CAF. We will expand on the remaining two features in our future efforts.

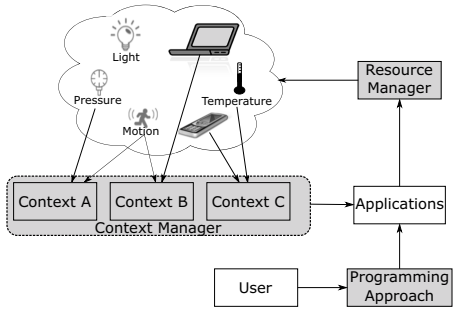

Fig. 2: Main features for Context-Aware Framework for IoT.

The main motivation behind CAF is to keep the user free from responsibilities other than expressing the application goal. Hence, the application written by the user should be independent of specific resources. On the other hand, the code must be expressed in a modular way, so that the resource manager can easily make decisions about usage of the available resources in the IoT. Such a modular and abstract approach will allow the system to move the code around the network. Hence, we propose a declarative programming approach for the user to write the application in self-contained blocks of code. A declarative approach allows the user to write meaningful statements instead of defining each action. These blocks of code can process only the values received as input(s) and send resulting output(s).

V. Proposal and Implementation of CAF

As discussed in Section III, the previous related work does not significantly contribute towards context-aware systems. However, there are some previous research and technological efforts which can be extended and adapted towards achieving a solution to accommodate context-aware systems. One of those prior efforts is T-Res [1].

T-Res provides a programming abstraction, specially tailored for the IoT paradigm. In T-Res, simple tasks can be created in the form of specified structures, called T-Res tasks. These tasks can be installed on IoT devices during runtime. This is done using CoAP procedures which allow configuring applications on a device and also the interaction among multiple devices. A T-Res task has four sub-resources. Input sources (*/is*) stores the devices used to collect input data for the task.

Output destination (*/od*) stores the devices used to deliver the output of the task. Processing function (*/pf*) stores the data processing to be done on the input data and compute output, hence the task itself. Last output (*/lo*) stores the most recent output from the task. Input and outputs are assigned to */is* and */od* by means of URI addresses. The application is provided as a compiled file to */pf*.

With such a structure T-Res establishes separation between data processing, input sources and output destinations. This is a required feature for building CAF, as our proposed model utilizes an abstraction to provide the user with the ability to write applications independent of the knowledge about each resource. T-Res comes close to provide similar abstraction between the application written by the user and the resources utilized by that application, which other approaches not provide. Next, we will take a closer look at T-Res with a simple example, and we will identify shortfalls in it and propose a solution to evolve it towards CAF.

A. Extending T-Res

To understand how T-Res works and how the proposed evolution towards CAF can be designed, we take a look at a simple application. The application maintains the temperature of a room between 19°C and 22°C. The room has two temperature sensors and one heating actuator. A T-Res task structure (see Figure 3) can be created for this application. One of the temperature sensors can be the host for the T-Res task. The sub-resources of T-Res task structure can be set to these resources. The */is* sub-resource can be the temperature sensor other than the host. The */od* sub-resource can be the heating actuator. The */pf* can be set to a script which takes the input from */is*, performs the calculation to check the temperature bound and provides the output instruction to */od*.

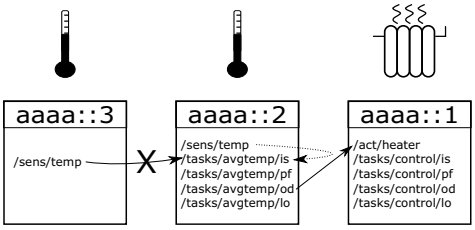

Fig. 3: T-Res task structure for the example.

In order to demonstrate a context change, assume the temperature sensor set as */is* sub-resource, is no longer available. This may happen due to energy failure, communication cost, or any other change in context. In this example we emphasize on the context of energy failure. Since there is another temperature sensor available in the room, the system should be able to detect this change, adapt to it and then use the other sensor as */is* sub-resource. However, in T-Res, that has to be done by the user by providing manual instructions using a CoAP agent. Although there may be many other possible solutions for this, we emphasize on extending T-Res because it provides similar abstraction required for the CAF.

As discussed in the example above, to adapt to the energy failure context, T-Res needs to detect the failure and change the resource allocated to appropriate task sub-resources. Our extension for mobility enables T-Res to do the same. We define Mobility as a feature where the resources and the processing function can be moved around to satisfy the context of the system, independent of user inputs. For example in above scenario, when the failure of temperature node is detected the system should be able to assign another available temperature node to */is* sub-resource of the task, as shown in Figure 3.

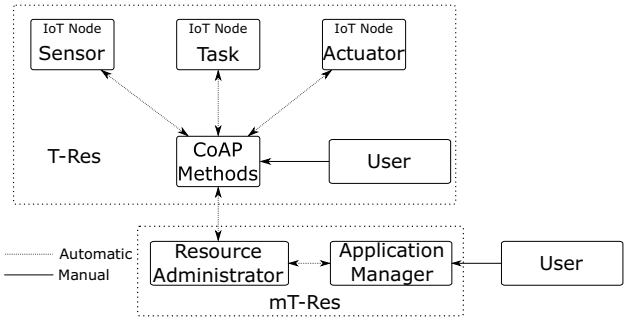

Fig. 4: mT-Res: Extension to T-Res.

We have designed and implemented an extension, called mT-Res [8], which facilitates above mentioned capabilities on top of T-Res. The extension is divided into two parts, *Resource Administrator* and *Application Manager*, as outlined in Figure 4. The *Resource Administrator* deploys the code to host devices, assigns the input and output devices and keeps track of any changes in the system.

The *Application Manager* inside mT-Res, which includes an application form where user can provide the task, keeps track of status of all applications with the help of the *Resource Administrator*. If any change in the resources is detected by the *Resource Administrator*, the *Application Manager* configures all applications again as per those changes.

The application form contains four fields, similar to the T-Res structure. These are input, output, host and code. In the code field, the user can provide the same code as required for T-Res. However, in the Input/Output/Host fields the user is asked to do an abstract selection from available resources, by selecting the type of resource the user wants to use for input or output or host for the code. The user does not need to provide URI addresses of specific resources. However, there can be more complex scenarios where more details are required from the user, such as spatial information, time bound, etc.

The *Resource Administrator* is enabled via python scripts, which provide automated CoAP operations (such as PULL, PUSH, GET, and OBSERVE), in form of python functions. The *Application Manager* is a django-based web framework, including the application form.

As the application is deployed and execution starts, the *Application Manager* updates the status of all active resources regularly. At any point if any operation returns with an error, the framework will once again execute the process to allocate resources. However, this time it will use another available resource for the corresponding error received earlier.

VI. Working Demonstration

We tested our implementation [8] on the Cooja simulator in the Contiki Operating System. Cooja provides emulated motes based on the MSP430 microcontroller such as WiSMotes. The simulation on Cooja is cycle accurate for each device and also

bit-level accurate for the radio transceivers of each device. This allows to have the same behavior in the simulator as on the actual hardware. IPv6 and CoAP support are provided by Contiki itself. Our implementation also provides support for the CoAP operations in python with the help of the txthings [9].

For our experiments, first we consider the same example as provided by T-Res. The simple example in T-Res has four WiSMotes as shown in Figure 5. The functions of each mote are as follows, mote 1 as border router; mote 2 as host sensor mote; mote 3 as input sensor mote and mote 4 as output actuator mote. Both sensor motes 2 and 3 can measure the same physical parameter. The host sensor mote 2 takes input from the sensor mote 3, divides the input values in half and provides output to the actuator mote 4.

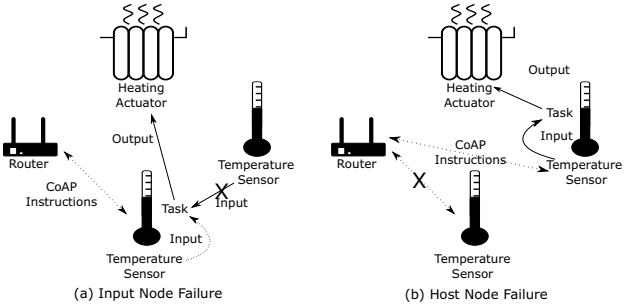

Fig. 5: Four motes with a simple T-Res application

In T-Res, these three devices have to be connected by a PUT request of CoAP. The compiled code of task is also deployed using another PUT request to the uri path of host mote 2. To complete the deployment, a POST request to host mote 2 is required. In T-Res the user is required to issue all these CoAP requests via the Copper CoAP [10] user agent for Firefox. In mT-Res, the user can provide the same code using the application form provided by the *Application Manager*. The *Resource Administrator* takes care of all CoAP operations.

In this example we take a look at a change in context due to energy failure. We demonstrate the actions of m-Tres on failure of two motes, host and input, respectively. First, let us assume that after some time of operation, input sensor mote 3 fails due to the context switch. The mT-Res will automatically reinitialize the deployment by substituting the mote 3 with mote 2. The *Resource Administrator* performs PUT request for input source as mote 2. After this, normal execution of the application resumes.

In a second case, host sensor mote 2 may fail instead of the input sensor mote 3. In that case, the mT-Res will reinitialize the deployment by substituting the mote 2 with mote 3 as host mote and assign itself as the input source as well. Once again, this would be done by *Resource Administrator*, which performs two PUT requests for both code and input respectively.

VII. Conclusion

We described mT-Res, an extension of T-Res, to enable mobility T-Res. mT-Res combines a django-based web framework for developing applications and autonomous CoAP operations. We also described a demonstration of mT-Res in Cooja simulations. In this paper, we targeted a controlled set of contexts such as energy failure or node failure. However, Mobility in CAF should be able to address many more contexts for complex scenarios. This is our future workplan as we describe next.

We intend to analyze the temporal performance of context-aware applications on mT-Res, such as response time and deadline guarantees. In order to measure the real-world performance, we will also evaluate the performance of mT-Res on implementations of IoT systems with heterogeneous devices.

mT-Res is the first step towards our CAF. There remain two other features of CAF that need to be tackled: the context manager to learn from entities in IoT and make decisions regarding contexts; and the resource manager to keep track of resources and (re-)deploy application with change in contexts. In the future, we intend to address complex scenarios with the integrated CAF.

Acknowledgments

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project FCOMP-01-0124-FEDER-020312 (Smartskin) and also by FCT/MEC and the EU ARTEMIS JU within project ARTEMIS/0004/2013 - JU grant nr. 621353(DEWI).

References

[1] D. Alessandrelli, M. Petraccay, and P. Pagano, "T-res: Enabling reconfigurable in-network processing in iot-based wsns," in *IEEE International Conference on Distributed Computing in Sensor Systems, 2013*.

[2] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *First Workshop on Mobile Computing Systems and Applications, 1994.*.

[3] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: Aiding the development of context-enabled applications," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, ACM, 1999.

[4] G. Fortino, R. Giannantonio, R. Gravina, P. Kuryloski, and R. Jafari, "Enabling effective programming and flexible management of efficient body sensor network applications," *Human-Machine Systems, IEEE Transactions on*, 2013.

[5] R. Newton, G. Morrisett, and M. Welsh, "The regiment macroprogramming system," in *Proceedings of the 6th international conference on Information processing in sensor networks*, ACM, 2007.

[6] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner, "The abstract task graph: A methodology for architecture-independent programming of networked sensor systems," in *Proceedings of the 2005 Workshop on End-to-end, Sense-and-respond Systems, Applications and Services*.

[7] C. Bormann, A. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *Internet Computing, IEEE*, vol. 16, no. 2, 2012.

[8] S. Gaur. (2015). T-res extension, [Online]. Available: https://bitbucket.org/shashankgaur_/tres_extension.

[9] M. Wasilak. (2015). Txthings, [Online]. Available: https://github.com/siskin/txThings/.

[10] M. Kovatsch, "Demo abstract: Humancoap interaction with copper," in *Proceedings of the 7th IEEE International Conference on Distributed Computing in Sensor Systems*, 2011.