CISTER
Research Center in
Real-Time & Embedded
Computing Systems

# Demo

## Automated resource allocation for T-Res

**Shashank Gaur**
**Raghuraman Rangarajan**
**Eduardo Tovar**

CISTER-TR-160209

# Automated resource allocation for T-Res

Shashank Gaur, Raghuraman Rangarajan, Eduardo Tovar

CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

## Abstract

This pap er presents a demo of an extension develop ed tosupp ort an existing programming abstraction for IoT: mTRes. mT-Res is an extension of the T-Res programmingabstraction, which allows users to write applications usinga web framework indep endent of resources. The pap er describ es an automated mechanism for allo cate resources tosuch applications and adapt to changes in those resources.

© CISTER Research Center
www.cister.isep.ipp.pt

# Demonstration Abstract: Automated resource allocation for T-Res

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

## ABSTRACT

This paper presents a demo of an extension developed to support an existing programming abstraction for IoT: mT-Res. mT-Res is an extension of the T-Res programming abstraction, which allows users to write applications using a web framework independent of resources. The paper describes an automated mechanism for allocate resources to such applications and adapt to changes in those resources.

## 1. INTRODUCTION

Programming abstractions have been a major focus for wireless sensor networks (WSNs). With the evolution of hardware and software technologies, WSNs have become an integral part of the Internet of Things (IoT). Such evolution allows IoT to leverage heterogeneous hardware available in the network to a much larger extent than simple WSNs. However, this increases complexity in designing programming abstractions for IoT. T-Res [1] is a recent important technological contribution in that direction.

Applications in T-Res are created through a structure called T-Res tasks. The T-Res task is the combination of four sub-resources: *Input Source (is)*, *Output Device (od)*, *Processing Function (pf)*, and *Last Output (lo)*. The main application logic is stored in the */pf*. The URI addresses of Input and Output devices are stored in */is* and */od* respectively. The most recent output of the application is stored in */lo*. The complete T-Res task can be hosted on an IoT device and Constraint Application Protocol (CoAP) procedures are used to configure the sub-resources. With such a structure T-Res establishes separation between data processing, input, and output. In the next section, we take a look at an example scenario to understand the need for our extension in T-Res.

## 2. MT-RES

Let us take a look at a simple application in T-Res. This application keeps the temperature of a room between a fixed range of 19°C and 22°C. Let us assume there are two temperature sensors and one heating actuator inside the room. To deploy this application using T-Res, a T-Res task has to be created, as shown in Figure 1. Initially, the T-Res task can be hosted on one of the temperature sensors. The */is* can be the temperature sensor other than the host. The */od* 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*.

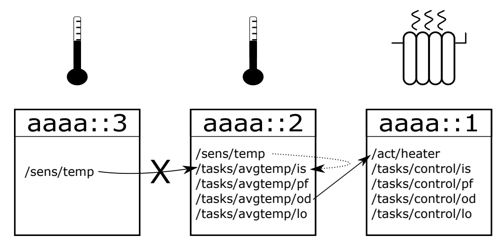


**Figure 1: T-Res example task structure**

Now assume after some time the temperature sensor set as */is*, is no longer available due to energy failure, communication cost, or any other change in the IoT. 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*. However, in T-Res, that has to be done by the user by providing manual instructions using a CoAP agent. We have designed and implemented an extension, called mT-Res [2], which facilitates above-mentioned capabilities on top of T-Res.

mT-Res is divided into two parts, *Resource Administrator* and *Application Manager*, as outlined in Figure 2. As of now, we keep both of these parts centralized in the system. The *Resource Administrator* deploys the code to host devices, assigns the input and output devices and keeps track of any changes in the system.

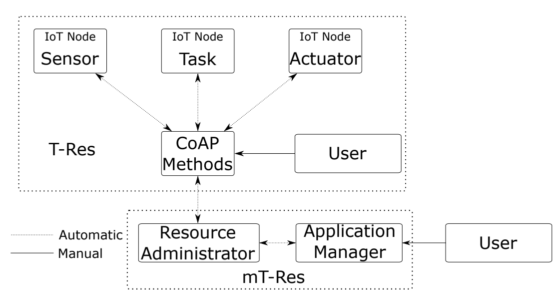


**Figure 2: mT-Res: Extension to T-Res**

For user to create a new application, the *Application Manager* provides a Django enabled web-interface. This interface 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. In the Input/Output/Host fields the user can do an abstract selection by selecting the type of resource the user wants to use. The user does not need to provide URI addresses of specific resources.

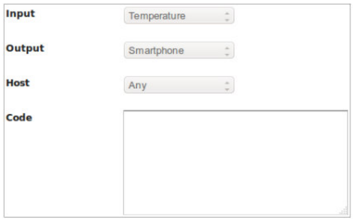


**Figure 3: Django enabled web-interface**

Once a user submits the application, the *Application Manager* utilizes *Resource Administrator* to check available resources and deploy the application accordingly. The *Application Manager* creates a list of acceptable resources for each allocation. If any of the lists are empty, the framework notifies users that the desired resources are not available. Next, it will choose one of the options from each list at random, assigns a category "Active" to selected resources and "Available" to remaining resources. Once the selections are available, it requests *Resource Administrator* to do the allocations.

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. Using these operations, *Resource Administrator* keeps track of all resources. If any operation returns with an error, the *Application Manager* will once again execute the process to allocate resources. However, this time, it will use another available resource for the corresponding error received earlier.

# 3. DEMO SETUP

We will demonstrate the mT-Res functionality using the Cooja simulator in Contiki operating system. We choose Cooja, because it provides emulated motes (WISMotes) and cycle accurate simulations for the MSP430 microcontroller. In addition, we can demonstrate radio transceivers with bit-level accuracy for the same devices. Also, IPv6 and CoAP support are provided inside Contiki as well. Source code for mT-Res is available at a public git repository[2].

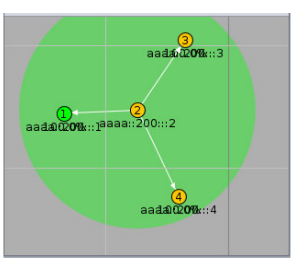

**Figure 4: Cooja Simulation**

For our demo, we will use the same example as provided by T-Res with four WiSMotes. The functions of each mote will be 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.

We will demonstrate the ability to submit the application using mT-Res for these motes. In mT-Res, the user will provide the same code as T-Res using the application form provided by the *Application Manager*. For host, input and output, the user will just define the type of motes to be selected. Since there will be two motes which can measure the same physical parameter, and mT-Res will perform autonomous resource allocation for that.

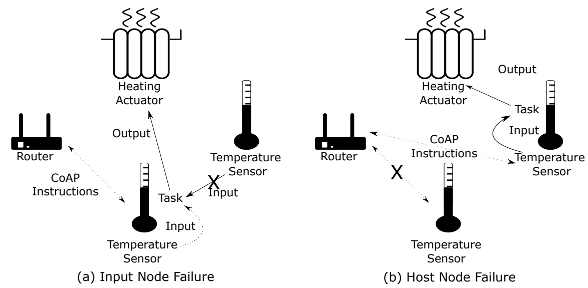


**Figure 5: Four motes with a simple T-Res task**

In our demo, the mT-Res will adapt to failure of two motes, host and input, respectively. When the input mote will fail, mT-Res will automatically re-allocate the input resource to the application by substituting input from the failed mote to host mote itself, which can measure the same physical parameter (see Figure 5(a)).

In the second case, failure of host mote will be detected. For that, the mT-Res will reinitialize the deployment by substituting the input mote as host mote and assign itself as the input source as well (see Figure 5(b)). This will demonstrate that mT-Res can also autonomously move the code from one device to another device.

# 4. CONCLUSION

In this demo, we extend capabilities of T-Res to provide autonomous resource allocations for IoT applications. In addition, mT-Res provides a web-interface for user(s) to input applications independent of specific resources. This extension is an effort to support context-aware IoT [3].

## 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, UID/CEC/04234/2013 (CISTER Unit); also by FCT/MEC and ERDF through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), 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).

# 5. 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] S. Gaur. mt-res. https://bitbucket.org/shashankgaur_/tres_extension, 2015.

[3] S. Gaur, R. Rangarajan, and E. Tovar. Extending t-res with mobility for context-aware iot. In *1st International Workshop on Interoperability, Integration, and Interconnection of Internet of Things Systems 2016.*