FUTEBOL UFMG
RCBTM Documentation

Fernanda Aparecida R. Silva[1]

[1]Computer Science Department (DCC) – Universidade Federal de Minas Gerais (UFMG)
Avenida Presidente Antônio Carlos, 6627 - Belo Horizonte - MG - Brasil

{fernandaars}@dcc.ufmg.br

1. Introduction

The UFMG testbed is managed by a software stack consisting of the RCBTM, the coordinator and the Aggregate Manager. The entry point to the testbed is the Aggregate Manager. When a user tries to allocate resources in the testbed, the information is sent to the Aggregate Manager, which then passes this information down the hierarchy. The purpose of this document is to describe the development and functioning of the RCBTM.

2. Software Description

The Restructured Cloud Based Testbed Manager is a new version of the Cloud Based Testbed Manager, software developed by the Trinity College Dublin with the purpose to manage a testbed that contain Virtual Machines and USRPs (Universal Software Radio Peripheral). The RCBTM follows the same principle, only providing support to more resources like Containers, Raspberries, Advanticsys Nodes and Switches Openflow. The CBTM was written in Python 2.7 and it is also possible to perform certain operation on the resources through it, such as start, shutdown and restart, when applicable. The same operations are possible in RCBTM, like the capacities of identifying if a node is currently in use, if it is booted and of retrieving its IP address. The term node will be used from here on to refer to an allocated resource on a host, e.g., a virtual machine allocated on a specific computer.

3. Diagrams

3.1. Class Diagram

The following image shows the simplified diagram class of the RCBTM, therefore the methods and attributes shown are a just except of the actual content of the classes. In the diagram, the class *Handler* works as the entry point of the system, receiving commands of external softwares - in this case, of the others softwares in the testbed - and dispatching them to the class CBTM. The *CBTM* class, in its turn, execute specific functions of allocation and deallocation, using the classes created as representations of each resource in the testbed. That classes - TelosB, VirtualMachine, USRP, RaspberryPI and etc - are based in a abstract class named *NodeCBTM* that has a object *user* of the class *userCBTM*, witch represents the user that are using that resource in a experiment. In parallel, the class *ResourceMonitorCBTM* verified continually the functioning of the equipments - with ping and ssh - and actualize the list of available resources list.

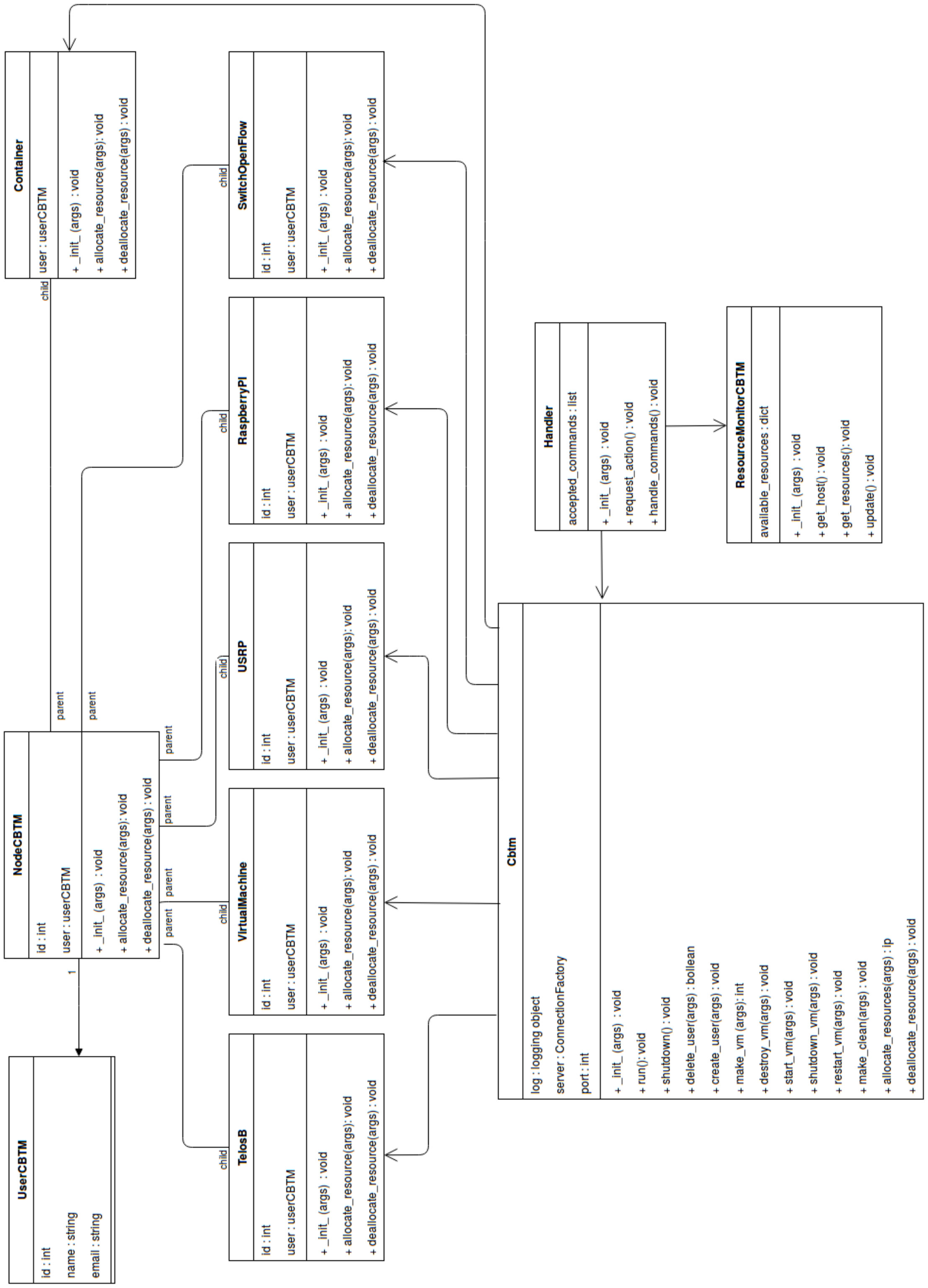
Container
user : userCBTM
+ _init_ (args) : void
+ allocate_resource(args): void
+ deallocate_resource(args) : void
SwitchOpenFlow
id : int
user : userCBTM
+ _init_ (args) : void
+ allocate_resource(args): void
+ deallocate_resource(args) : void
RaspberryPI
id : int
user : userCBTM
+ _init_ (args) : void
+ allocate_resource(args): void
+ deallocate_resource(args) : void
USRP
id : int
user : userCBTM
+ _init_ (args) : void
+ allocate_resource(args): void
+ deallocate_resource(args) : void
VirtualMachine
id : int
user : userCBTM
+ _init_ (args) : void
+ allocate_resource(args): void
+ deallocate_resource(args) : void
TelosB
id : int
user : userCBTM
+ _init_ (args) : void
+ allocate_resource(args): void
+ deallocate_resource(args) : void
NodeCBTM
id : int
user : userCBTM
+ _init_ (args) : void
+ allocate_resource(args): void
+ deallocate_resource(args) : void
UserCBTM
id : int
name : string
email : string
Handler
accepted_commands : list
+ _init_ (args) : void
+ request_action() : void
+ handle_commands() : void
ResourceMonitorCBTM
available_resources : dict
+ _init_ (args) : void
+ get_host() : void
+ get_resources(): void
+ update() : void
Cbtm
log : logging object
server : ConnectionFactory
port : int
+ _init_ (args) : void
+ run(): void
+ shutdown() : void
+ delete_user(args) : boolean
+ create_user(args) : void
+ make_vm (args): int
+ destroy_vm(args) : void
+ start_vm(args) : void
+ shutdown_vm(args) : void
+ restart_vm(args) : void
+ make_clean(args) : void
+ allocate_resources(args) : ip
+ deallocate_resource(args) : void
child
parent
child
parent
child
parent
child
parent
child
parent
child
parent
1

Figure 1. RCBTM - Class Diagram Overview