FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Web Based Simulation for RAHYMS

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software und Information Engineering

eingereicht von

Mehmetcan Burak Karaoglan
Matrikelnummer 0825659

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.Doz. Dr. Hong-Linh Truong
Mitwirkung: Ph.D. Muhammad Zuhri Catur Candra

Wien, 1. August 2016

Mehmetcan Burak Karaoglan | Hong-Linh Truong

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Web Based Simulation for RAHYMS

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Mehmetcan Burak Karaoglan
Registration Number 0825659

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Priv.Doz. Dr. Hong-Linh Truong
Assistance: Ph.D. Muhammad Zuhri Catur Candra

Vienna, 1st August, 2016

Mehmetcan Burak Karaoglan	Hong-Linh Truong

Erklärung zur Verfassung der Arbeit

Mehmetcan Burak Karaoglan
Vienna, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. August 2016

Mehmetcan Burak Karaoglan

Acknowledgements

First and foremost, I have to thank my family - my parents and my siblings for standing beside me.

I would also like to show gratitude to my research supervisors, Priv.Doz. Dr. Hong-Linh Truong and Ph.D. Muhammad Zuhri Catur Candra. Without their assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished. I would like to thank you very much for your support and understanding.

Abstract

This thesis analyzes the existing system of RAHYMS(Runtime and Analytics for Hybrid Computing Systems). RAHYMS analyzes hybrid computing systems, which intertwine humans and machines in Internet-scale, both as active problem solvers.

This thesis presents a graphical user interface for simulation mode. The simulation mode, allows consumers to simulate a pool of compute units and a series of task requests with customizable configurations.

In this thesis we provide an interface that integrates with the existing simulation mode. It is recommended to first study RAHYMS before analyzing this thesis. This platform is open-source and available on GitHub[1]. [1]

[1] https://github.com/tuwiendsg/RAHYMS

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Overview

RAHYMS analyzes the systems as *Hybrid Human-Machine Computing Systems* (HCSs) employing humans and machines as computing units, where tasks are shared by humans and machines. The simulation of an HCS consist of two phases: first phase is initial phase, where compute units are generated with configurable initial properties and second phase is execution phase where tasks are generated, and for each task a compute units collective is created to execute the task. The execution phase consists of cycles. In every cycle, the task generator configurations are processed to generate tasks. After a configured number of cycles have passed, the task generation stops, and simulation is finished once all the remaining running tasks are completed. Our graphical user interaction system of simulation provides an interface to users where compute units and tasks configured, simulation result graphically analyzed.

1.2 Thesis Structure

This thesis is organized as follows:

- *Chapter 2* focuses on inconveniences of existing simulation of SCU. Also presents our motivating scenario.
- *Chapter 3* discusses the existing works on graphical interface of simulation of humans.
- *Chapter 4* presents our graphical user interface of Simulation-Mode.

CHAPTER 2

Requirements

In this chapter we will be discussing about the previous problems we faced without our graphical user interface of Simulation-Mode. Hence, we will be presenting our motivations to solve such problems.

2.1 Motivating Scenario

In the existing simulation system of RAHYMS, for each simulation the application needs to be rerun each time. And this causes time loss and inconvenience for user. Also the simulation tasks, the simulation units, simulation-metric-information and simulation results information are not able to persist into database. In this case when user would want to access to the mentioned information, it is simply not possible. Due to absence of a graphical user interface it is difficult to analyze the system and start the simulation accordingly. We provide user a simpler interface, and this also provides convenience for those who have no programming skills. Lastly in the existing system the created content of simulation result is not graphically shown causing user a complex pile of information. However in our analytic interface we provide user a graph for metric correlation between those parameters saved in simulation result.

2.2 Chapter Summary

In this chapter we touched upon inconveniences of existing simulation system of RAHYMS and we compared it to our system and to see how our system provides user save on time, and work.

CHAPTER 3

State of the art

In this chapter, we discuss the present state of the art in the area of graphical user interface tools for defining SCU and simulation of humans.

3.1 Related Work in Simulation of Humans

There have not been made many related work to support graphical SCU simulation, except for the simulation engine RAHYMS, which is founding work of this thesis that we introduced in the introduction section.

CHAPTER 4

Simulation Mode UI - HCU Web Based Simulation

The UI of *Runtime and Analytics for Hybrid Computing Systems* RAHYMS consists of mainly two parts; Interactive-Mode and Simulation-Mode. Interactive mode allows consumers to submit task requests to be executed by a provisioned compute units collective. This thesis' aim is to implement a convenient graphical user interface for Simulation-Mode. The architecture of UI can be studied in Figure 4.1 Graphical User Interface of Simulation Mode consists of mainly four parts. The Figure 4.2 shows us those mentioned parts.

First part is the simulation-unit, in which user can create a new simulation unit and update an existing one. Second part is the simulation-task, in which user can create a new simulation task and update an existing one.

In third part user can specify the simulation parameters such as algorithm properties, simulation name and description etc. And user can determine which units and tasks should be selected to start the simulation. After simulation executes, user can navigate to the fourth part which is Analytics. This part provides the analytics of the simulation, the metric correlation between two parameters can be analyzed on the Analytics tab.

In this graphical user interface the pencil icons represent an edit functionality, the trash icons represent delete functionality and lastly the plus icons represent create functionality.

4.1 Simulation Unit

As it is shown in Fig. 4.2, the first thing that appears in Unit Generator tab is the table of all available units. Three units added as default to the system so that it may provide convenience. Desired and detailed information available in Appendix A.1.1. If need be, these units can be changed or deleted to add new ones.

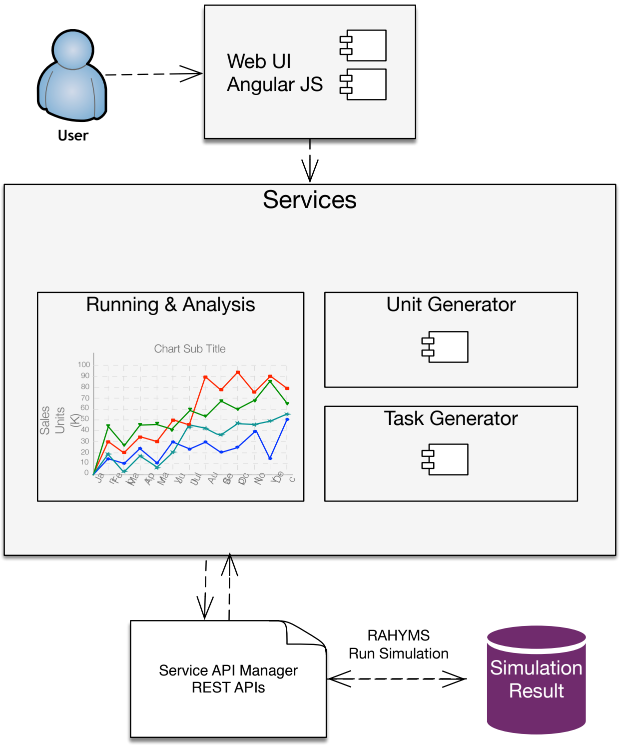

Figure 4.1: Architecture Design of the UI

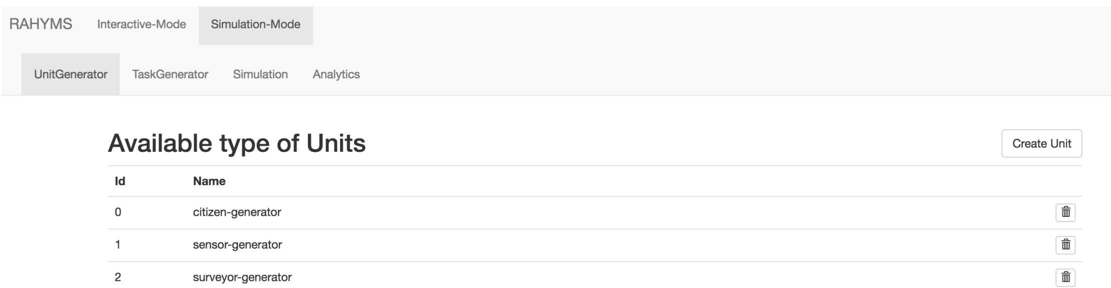

Figure 4.2: Unit Generator Table

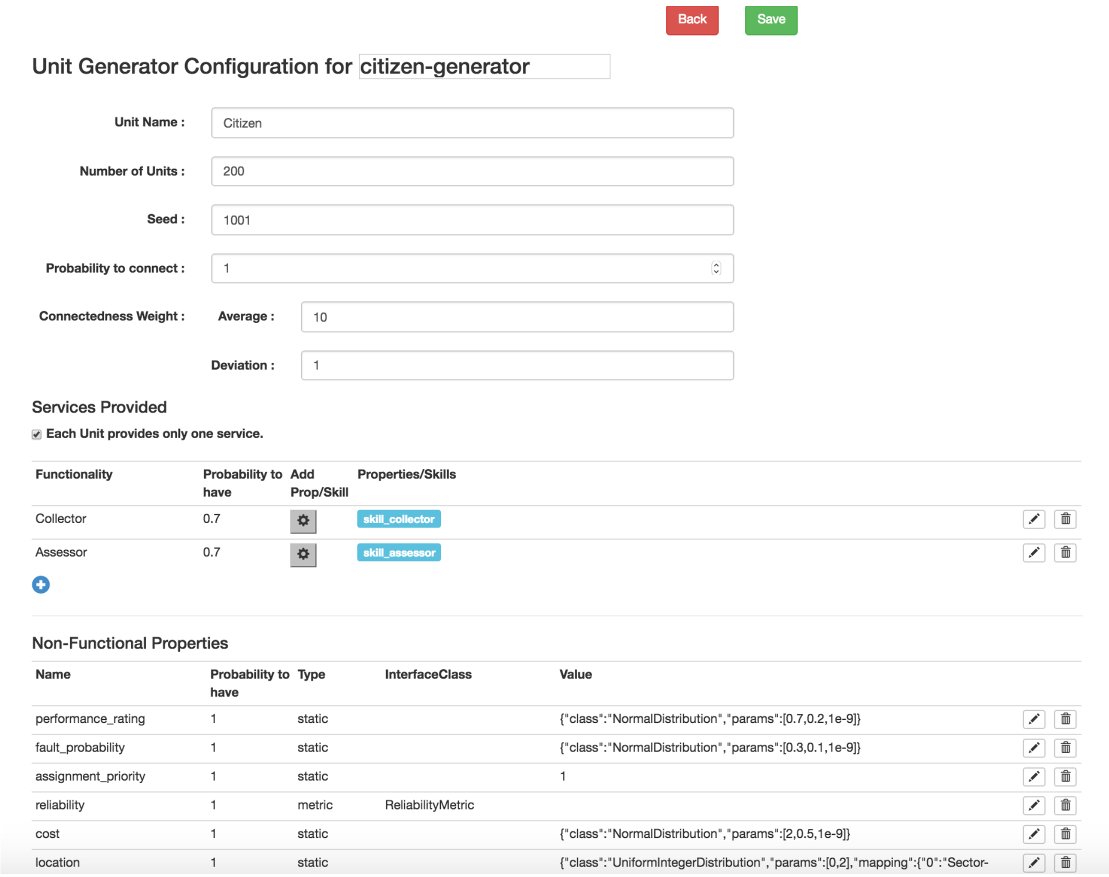

Figure 4.3: Unit Generator Detail

In Unit Generator tab on the right side, by using the create unit button one can create a new unit, and be mindful of given name to be unique. After creation, the system receives the added properties from first unit that is embedded in database unit section with table name ''SIMULATION_UNIT'' and instantiate this added unit object with these properties as default. If desired, these properties can be changed by accessing unit details, which we are going to explain in detail on next paragraphs. On available type of units table, on the right side of each unit's row, there is a function, shaped as trash bin, for deleting such unit. By using this function one can delete any desired unit. However, this process cannot be undone.

The detail page of unit generator can be accessed by clicking on each column of units. And one can see the detail page which shown in Fig. 4.3. This section in general consists of 3 parts; general unit properties, provided service properties and non-functional properties.

In Fig. 4.4, we exemplify a compute units generator configuration annotated to describe the purpose of the configuration. This example shows a generation of citizens as compute units. The two numbers in *<normal-distribution-config>* define how a value should

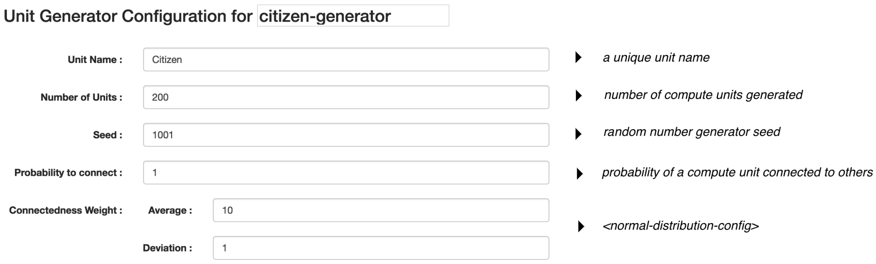

Figure 4.4: Unit Properties

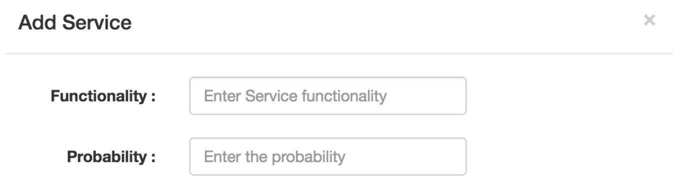

Figure 4.5: Unit Service Add Modal

be populated with a random number generator, the average, also known as mean, and stardard deviation are the properties of normal random generator. The last property of normal distribution config is inverse cumulative distribution accuracy, and this is given by default with value $[1.0E - 9]$.

Services are provided by each generated compute unit. A new service can be created by clicking plus icon as shown in Fig. 4.5. Functionality defines the name of functionality e.g., "DataCollection" and must be unique, the probability is the chance that the compute unit has this functionality and both parameters must be given.

To edit the service functionality and probability we provided an edit functionality, which is the pencil icon at the end of each corresponding row of service.

A new property/skill can be created by clicking gear icon. If one of the properties clicked, then new modal opens, where that property can be edited and also delete functionality will be activated, so that the property/skill can be also removed from service, which is shown in Fig. 4.6 with *<delete-functionality>*.

Value Generator, which is similar to *Connectedness Weight* used in Section 4.1 is shown in Fig. 4.4. This random generator also used in several sections such as `unit non-functional property add`, `task functional constraint add and lastly task non-functional property add`. But the class distribution could be different than normal distribution. Uniform Integer Distribution and static are other options. These distribution classes are available from `Apache Common Math package org.apache.commor`

[1]https://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/distribution/package-summary.html

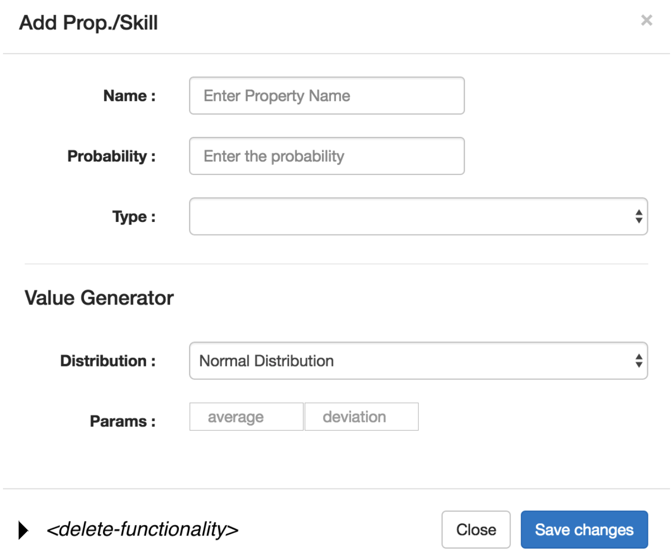

Figure 4.6: Unit Service Property/Skill Add Modal

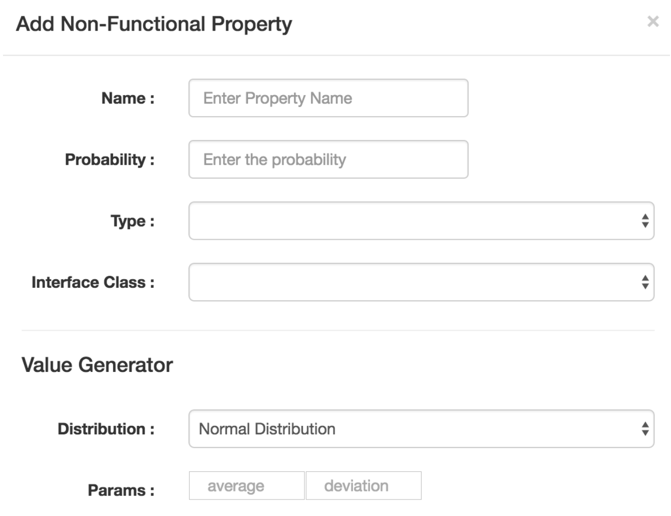

Figure 4.7: Unit Non Functional Add Modal

The optional mapping entry defines a mapping from an integer number distribution to a certain value, e.g., a string value, which will be activated if *UniformIntegerDistribution* selected. Those values can be extended if needed, details available in Appendix A.1.2.

Non functional properties is also shown in Fig. 4.7. Functionality-specific properties also known as properties of a service and non-functional properties have the values; name stands for the property name e.g., "location" , "cost", etc., probability is the chance of compute unit having this property, type can be “metric”, “skill”, or “static”, if needed additional types can be added into UI. Details available in Appendix A.1.2. A property can be of three types: metric property, which defines a property its values can be retrieved externally, skill property, which defines the functional capability of a human-based compute units, and static is for all other properties (note that despite of the name, the static property value can still be modified). For metric property, the interfaceClass entry defines the class implementing MetricInterface that provides the value of the property.

After all settings configured in order to persist the changes, save button must be clicked. Be mindful if the page is closed without save button clicked any unsaved changes will be lost.

4.2 Simulation Task

On Task Generator tab, one sees the same design as it is in unit part shown in Fig. 4.2, all basic functions such as create task, delete task and edit task are also same as in unit here. As well as in unit, two default tasks are added into database, in order to provide convenience for user. Desired and detailed information is available in Appendix A.1.1. Also in here after creation of task, the system receives the added properties from database task section with table name ‘’SIMULATION_TASK” and instantiate this added task object with these properties as default.

When accessed to the detail page, the Fig. 4.8 appears. Detail page consists of three main parts; general task properties, role requirements and lastly non-functional constraints as they are shown in Figure 4.8.

Taks Occurance is an optional value, which defines the number of tasks generated at each cycle. Task load needed for how long the task will be executed by a unit. And this values are similar to *"Connectedness Weight"* that is used in unit generator (Section 4.1 with Fig. 4.4).

In *"Task Role Requirement Add Modal"*, the roles that are needed for this task are determined. Functionality defines a functional requirement for the role e.g., *"DataCollection"*. *Relative Load Ratio* which is defined with (effective load = relative load *task load). *"Depends on"* section which is shown in Fig. 4.9 determines the dependability of added role towards the other roles that are embedded into that unit, this dependability can be strong or normal, if desired it can be activated.

A new functional constraint can be created by clicking gear icon. If one of functional constraint clicked, then new modal opens, in which that constraint can be edited. And with *<delete-functionality>* annotated in Figure 4.10 the constraint could be removed from role.

The Comparator Class is a fully-qualified name of a class implementing the *java.util.Comparator* interface. Several comparator classes are provided in *at.ac.tuwien.dsg.hcu.common.sla.comparator*

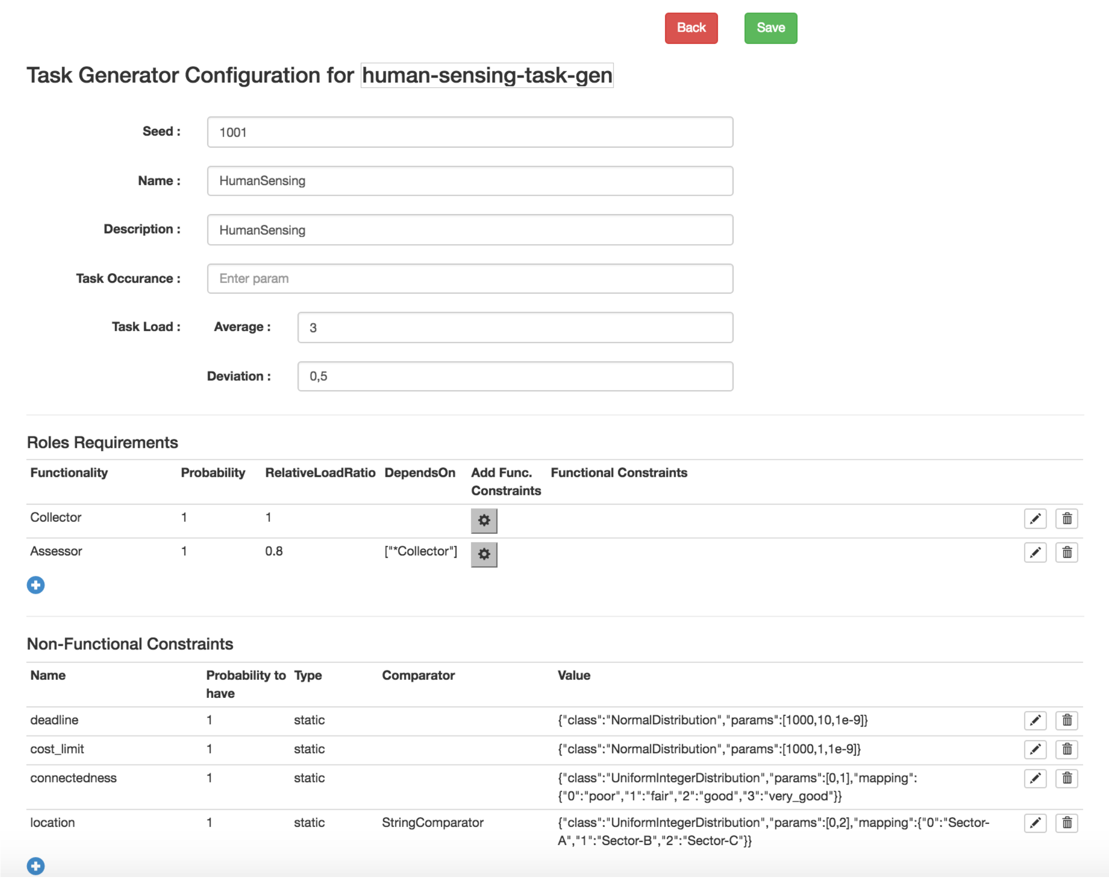

Figure 4.8: Task Generator Detail

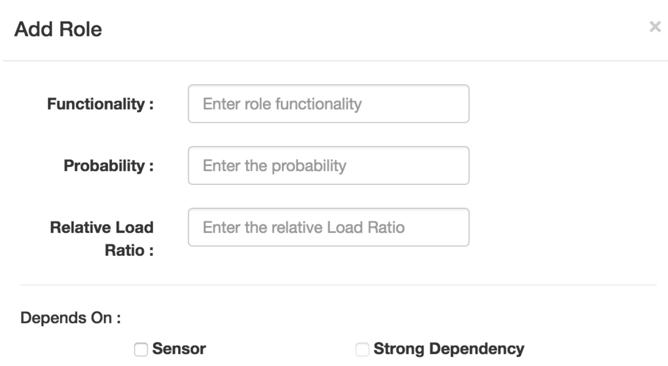

Figure 4.9: Task Role Requirement Add Modal

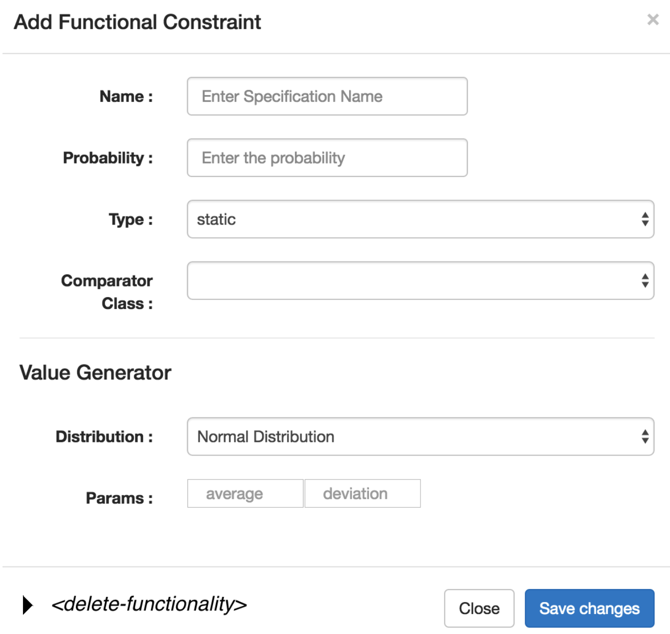

Figure 4.10: Task Functional Constraint Add Modal

package: StringComparator, NumericAscendingComparator, NumericDescendingComparator, and FuzzyComparator. If needed these values can be extended, details available in Appendix A.1.3.

In non functional constraints part all desired non-functional properties are determined, which is similar to the Fig. 4.10.

4.3 Simulation Algorithm Properties

Here we provided algorithm properties for simulation. After determining units and tasks in Sections 4.1 and 4.2 that are added to the system, they can be dragged and dropped to the side boxes, shown in Fig. 4.11. Lastly when the algorithm properties are set, the simulation is ready to be executed. While simulation is being executed, the algorithm first generates compute units, secondly tasks, and lastly creates for each task a compute units collective to execute the task. The second phase consists of cycles. In every cycle, the task generator configurations are processed to generate tasks. After a configured number of cycles passed, the task generation stops, and simulation is finished once all the remaining running tasks are completed.

Detailed algorithm properties become activated once the *AntColonyOptimization* for the algorithm is selected (Fig. 4.12). In *aco variant* section the aco variant properties vary depending on which aco variant chosen. (Fig. 4.13). Currently, available formation algorithms are:

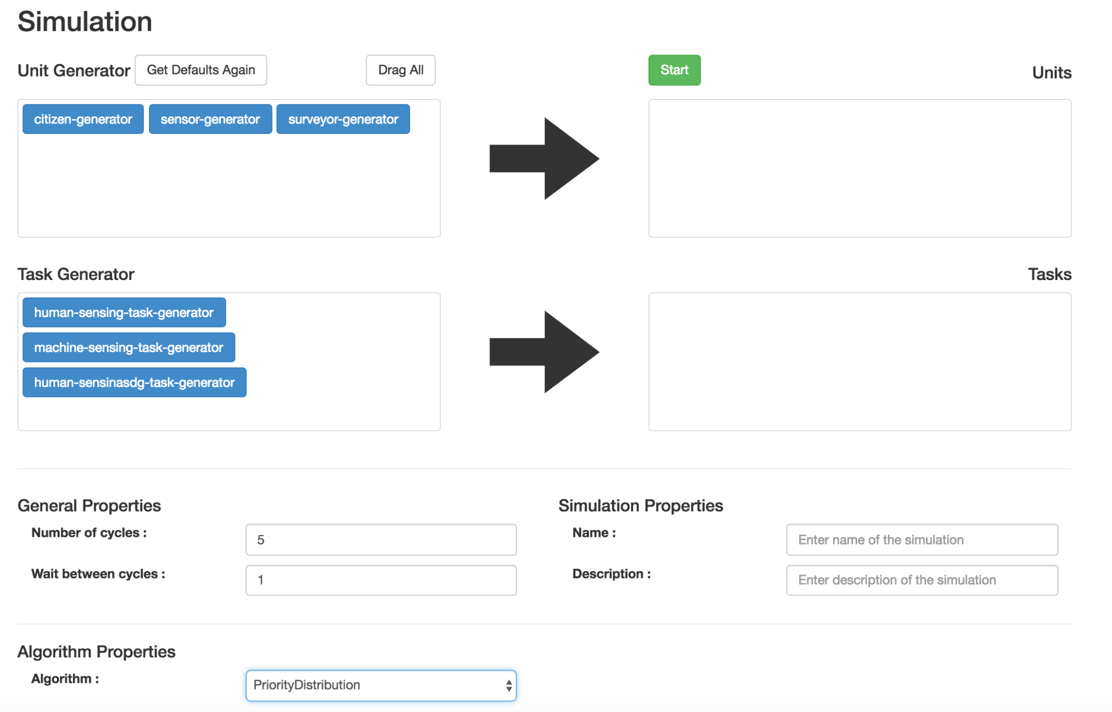

Figure 4.11: Simulation Properties Detail

- `FairDistribution` algorithm, which distributes tasks uniformly to all qualified compute units,
- `PriorityDistribution` algorithm, which distributes tasks based on the priority of each compute unit specified in `assignment_priority` property, e.g., a compute unit with priority equals to 2 has twice probability to be assigned to tasks compared to compute units with priority equals to 1,
- `EarliestResponse` algorithm, which assigns tasks to compute units with the earliest estimated response time (e.g., the *first come first serve* strategy),
- `GreedyBestFitness` algorithm is a greedy heuristic strategy, which processes each task role iteratively, and for each role a compute unit with the best local fitness value is selected,
- `GreedyHillClimbing` algorithm finds an initial solution similarly as the `Greedy BestFitness` algorithm, and refines the solution further using a *hill climbing* technique, the number of cycles for hill climbing is specified using `maximum_number_of_cycles` parameter,
- `ACOAlgorithm` algorithms, which find the best solution using Ant Colony Optimization. Currently the following variants of ACO algorithms are supported:

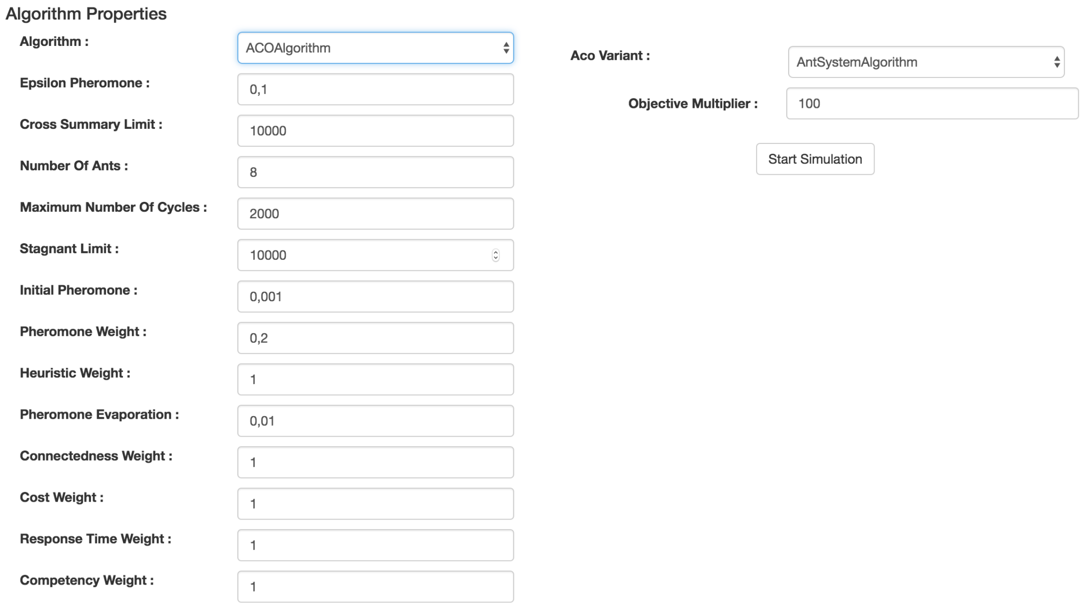

Figure 4.12: Simulation Ant Colony Optimization Algorithm Properties

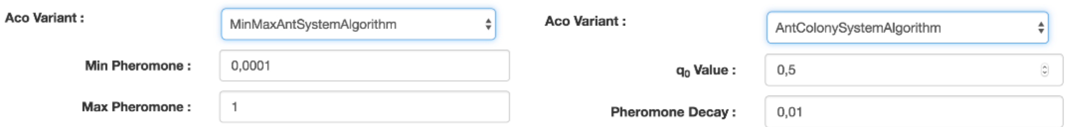

Figure 4.13: ACO Variant MinMaxAntSystem & AntColonySystem Algorithm Variants

`AntSystemAlgorithm`, `MinMaxAntSystemAlgorithm`, and `AntColonySystemAlgorithm`.

If needed another type of algorithms can be added, details available in Appendix A.1.4. To prevent confusing, the *ACOAlgorithm* properties are set with default values if need be those can be changed.

Once all the properties are set, we are able to click on start button, we could just wait for simulation to be completed or it is also possible to navigate analytics tab by clicking on opened wait modal.

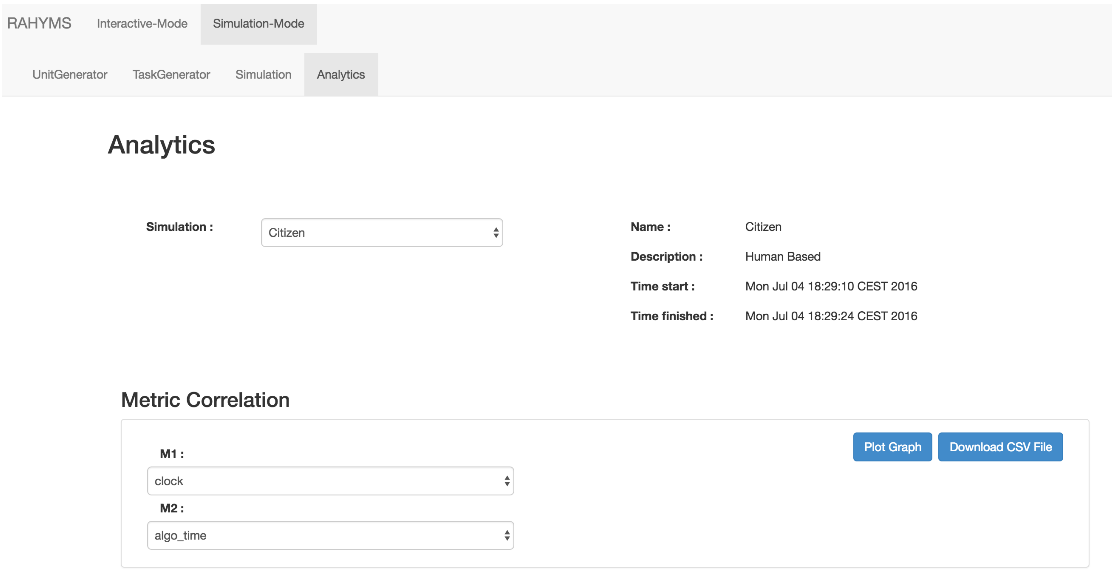

Figure 4.14: Analytics Tab Detail

4.4 Analytics

This part assists us to analyze the simulation results. Once simulation is finished simulation results are saved as `CSV`[2] file, all information are called accordingly to metric components that were selected and the relation between the two are drawn in to a graph. This way our system provides user an easy way of visual analyzing. On the page, a saved, already running simulation can be chosen, afterwards according to the chosen simulation, on the right side a detailed section can be seen as it is shown in the Fig. 4.14.

If the chosen simulation hadn't been completed yet, the metric correlation section won't be shown. Simulation must be completed first in order to perform a transaction. Only couple of information about the simulation are provided by system, which are the name of the simulation and the description of the simulation. To see whether the simulation completed or not, one can watch timeFinished attribute and follow the process by clicking ''refresh" button. Running means it is not yet completed. (Fig. 4.16)

After the simulation is completed, mectric correlation section will be shown and graph can be drawn by choosing metrics. First metric represents ''X" axis and consists of *"clock"* and *"task_id"* attributes. If "clock" for x axis selected then a scatter chart will be drawn, otherwise line chart. Second metric represents ''Y" axis and consists of simulation result values which they are available in Appendix A.1.5. As shown in the Figure 4.15, a plot graph can be drawn after choosing metrics. If desired, the CSV file (shown in Fig. A.1) can be downloaded and this file holds the simulation results.

[2] CSV stands for comma separated value

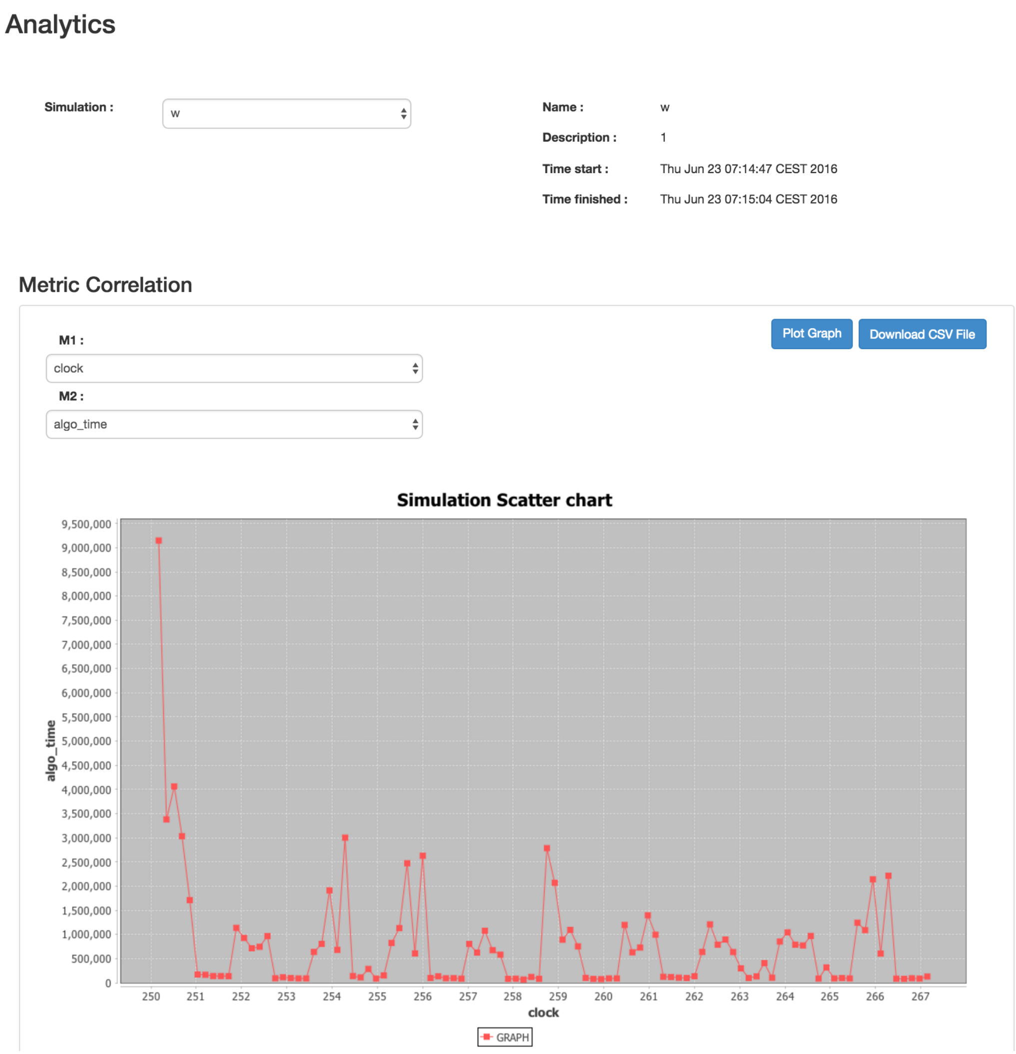

Figure 4.15: Analytics Tab - Metric Correlation Graph

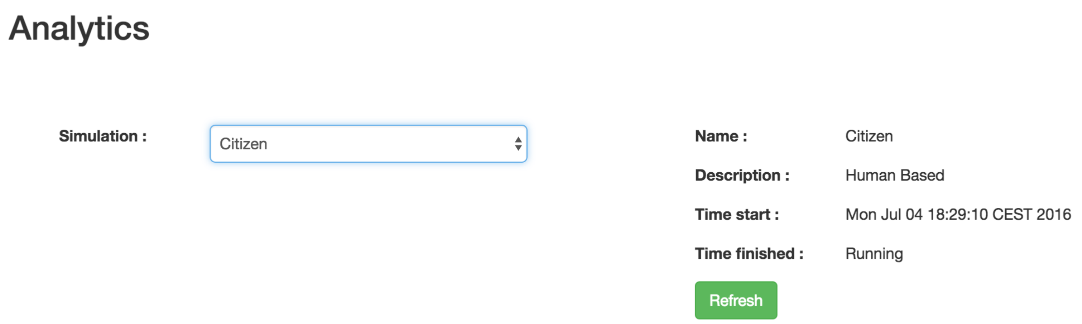

Figure 4.16: Analytic - Simulation Detail Properties

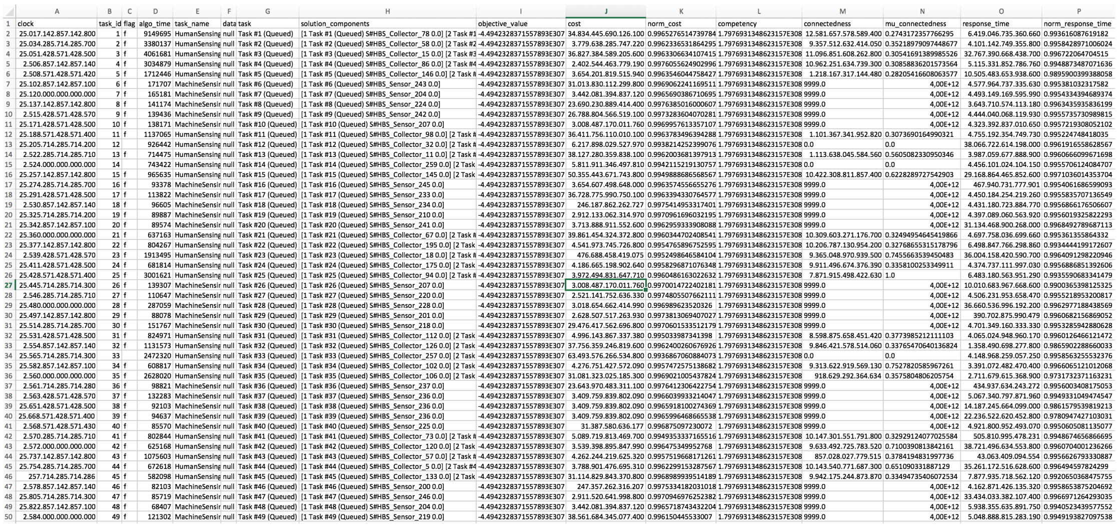

Figure 4.17: CSV File - Simulation Result Properties

The scatter and line charts are provided by our system, to visualize the simulation result. Right now only one analytic implemented (Metric correlation between two metrics). If desired a new analytic can be added to system, details available in chapter Appendix A.1.5.

4.5 Chapter Summary

In this chapter we studied the graphical user interface of Simulation-Mode, and gave detailed information about how the system works. The connection between all parts of Simulation-Mode have been described and also all each part, UnitGenerator, TaskGenerator, Simulation and Analytic examined separately.

APPENDIX A

Prototype Documentation

This appendix provides a documentation of the simulation platform Runtime and Analytics for Hybrid Computing Systems (RAHYMS).

In this appendix, first we give a detailed information how some properties can be extended and next the details of the operation of the platform, i.e. how to use the Application Programming Interfaces (APIs).

A.1 Extendable Properties Documentation

A.1.1 Getting Started

This section provides a documentation of the properties, which can be extended with new values. While simulation web application is being setup, some services and some task generators have been saved in to the database to provide a better understanding, convenience and for usage of simulation task generator and simulation unit generator. The mentioned default tasks and units functions can be found in class *at.ac.tuwien.dsg.hcu.common.util.DefaultMongoData* and the properties file in *at.ac.tuwien.dsg.hcu.config.simulation-web-default.properties* contains needed information.

In this project *MongoDatabase*[1] is used to persist the simulation properties. The main properties of *MongoDatabase* can be found in *at.ac.tuwien.dsg.hcu.config.simulation.properties*. Database connection provided by *mLab*[2].

A.1.2 Unit Generator Extendable Properties

[1] https://docs.mongodb.com/manual/
[2] https://mlab.com/

Value Generator Distribution properties are taken from file *at.ac.tuwien.dsg.hcu.hcu-rest.src.main.resource.webui.extend.unit-extend.json* with key `<distribution>`, those values can be changed or added new ones. But be mindful that the system has to cooperate with the new extended distribution in order to complete simulation. If the system does not support the new distribution, the needed changes should be implemented into system.

Functional & Non-Functional Property Types are available with keys `<functional-property` and `<non-functional-property-type>`. How to find the file and the possible reflexions of simulation algorithm is already explained above Paragraph A.1.2.

A.1.3 Task Generator Extendable Properties

All the task generator extendable properties are available in the same file path of above described `unit-extend.json` A.1.2. But the file name is `task-extend.json`.

Functional & Non-Functional Constraint Types are available with keys `<functional-constr` and `<non-functional-constraint-type>`. How to find the file and the possible reflexions of simulation algorithm is already explained above Paragraph A.1.2.

Comparator Class properties defined with key `<comparator-class>`.

A.1.4 Simulation Extendable Properties

The path of the simulation extendable properties file is the same as `unit-extend.json` A.1.2. But the file name is `simulation-extend.json`

Algorithms are available with key `<algorithm>`. A new algorithm can be added into array.

Aco Variant properties are available with key `<aco-variant>`. A new aco vairant can be added into array.

A.1.5 Analytics Extendable Properties

The path of the analytics extendable properties file is the same as `unit-extend.json` A.1.2. But the file name is `analytics-extend.json`

Metric Correlation - Metric values are available with keys `<metric-m1>` and `<metric-m2>`. `"m1"` stands for first metric and m2 stands for second metric, if needed the values of them can be changed or extended.

For Additional Analytic Implementation Simulation result information saved in table `<simulation-graph-information>` in database. For a new analytic implementation the result information will be needed. These information are available in method `<getColumnForGraph>` of *at.ac.tuwien.dsg.hcu.common.util.MongoDatabase* class. Two parameters are needed for this method: first one is "columnName", which determines desired column of result information and the second one "simulationId" determines from which simulation result should be returned. *at.ac.tuwien.dsg.hcu.simulation.util.SimulationGraphDrawer* makes the drawing of the graphs by using *org.jfree.chart.JFreeChart*[3]. Based on given "xAxis", "yAxis" and "simulationId" it draws the graph as image accordingly.

A.2 Application Programming Interface

The Application Programming Interface provided by our platform is a RESTful API, which provides CRUD (create, read, update, delete) operations for three entities: `simulation-task`, `simulation-unit` and `simulation`.

A list of applicable Operations for all APIs are following:

Request URL prefix
: http://<SERVER_HOST>:<SERVER_PORT>/<REST_CONTEXT_PATH>/api
: default: `http://localhost:8080/rest/api`

`POST` and `PUT` parameters encoding (in the request body)
: `application/x-www-form-urlencoded`

HTTP response codes
: `200`: Successful
: `201`: Created successfully
: `404`: Error, entity not found
: `409`: Error, entity already exists

Response body encoding
: `application/json`

The `simulation-task`, `simulation-unit`, and `simulation` entities and their API operations are described as follows. Documentation of API operations for the entities can be also viewed online from the Swagger API playground provided by the platform. When an API accepts a URL path parameter, it is shown here inside curly brackets. Actual request should not include the brackets in the URL.

```
{
    "seed": 1001,     ► random number generator seed
    "taskTypes": [    ► list of task types that should be generated
        {
```

[3]http://www.jfree.org/jfreechart/

```
            "name": "HumanSensingTask",
            "description": "An explanation of the task",
            "tasksOccurance": {*\textit{<distribution-config>}*},  ► number of tasks
                generated at each cycle
            "load": {<distribution-config>}, ► to simulate how long the task will be
                executed by a unit
            "roles": [  ► list of roles for the task
                {
                    "functionality": "DataCollection",  ► a functional requirement for
                        the role
                    "probabilityToHave": 1.0,   ► probability the role has this
                        functional requirement
                    "relativeLoadRatio": 1.0,   ► effective load = relative load *task
                        load
                    "dependsOn": ["...", ...],  ► a list of role functionality that
                        this role depends on (collective dependency)
                    "specification": [  ► role-level non-functional constraints
                        <specification-config>,
                        ...
                    ]
                },
                ...
            ],
            "specification": [  ► task-level non-functional constraints
                <specification-config>,
                ...
            ]
        },
        ...  ► multiple task types can be defined
    ]
}
```

Listing A.1: An Example of Task Generator Configuration

Operations on `simulation-task`

a) **`GET /simulation-task`** ► *List all simulation tasks*

Response body on success:

```
[
{
"id": 576b7471040bc30d1088fdb3,
"name": "...",
"task": "..."
},
...
]
```

Note:

- `id` is an auto-generated unique object id of the simulation task
- `task` is the string converted from json object described above Listing A.1

b) **`GET /simulation-task/{objectId}`** ► *find a simulation task specified by objectId*

Response body on success:

```
{
"id": 576b7471040bc30d1088fdb3,
"name": "...",
"task": "...",
}
```

Note: refer to note for `GET /simulation-task`

c) **`GET /simulation-task/default`** ► *find default simulation-task, which is first simulation-task stored in database*
Response body on success:

```
{
"id": 576b7471040bc30d1088fdb3,
"name": "...",
"task": "...",
}
```

Note: refer to note for `GET /simulation-task`

d) **`POST /simulation-task`** ► *Submit a new simulation task request*
Parameters:
- `name` (String): Simulation Task's name
- `task` (String): Simulation Task's content description described above in `GET /simulation-task`

Response body on success: refer to response body for `GET /simulation-task`

e) **`PUT /simulation-task/{objectId}`** ► *Update an existing simulation task specified by objectId*
Parameters:
- `objectId` (String): Simulation Task's objectId to be updated
- `name` (String): Simulation Task's name
- `task` (String): Simulation Task's content description described in `GET /simulation-task`

Response body on success: refer to response body for `POST /simulation-task`

f) **`DELETE /simulation-task/{objectId}`** ► *delete an existing simulation task specified by objectId*
Response body on success: None

Operations on `simulation-unit`

```
{
  "seed":1001,    ► random number generator seed
  "numberOfElements":200,    ► number of compute units generated
  "namePrefix":"Citizen",
  "connection":{
    "probabilityToConnect":0.4,  ► probabiity of a compute unit connected to others
    "weight":<distribution-config>
  },
  "services":[    ► the functional services provided by each generated compute unit
    {
      "functionality":"DataCollection",
      "probabilityToHave":0.7,    ► probability the compute unit has this
          functionality
      "properties":[    ► functionality-specific properties
        <property-config>,
        ...
      ]
    },
    ...
  ],
```

```
  "commonProperties":[    ► non-functional properties
    <property-config>,
    ...
  ]
}
```

Listing A.2: An Example of Compute Units Generator Configuration

a) **`GET /simulation-unit`** ► *List all simulation units*
Response body on success:

```
[
{
"id": 576b7471040bc30d1088fdb3,
"name": "...",
"unit": "..."
},
...
]
```

Note:
- `id` is an auto-generated unique object id of the simulation unit
- `unit` is the string converted from json object described above Listing A.2

b) **`GET /simulation-unit/{objectId}`** ► *find a simulation unit specified by objectId*
Response body on success:

```
{
"id": 576b7471040bc30d1088fdb3,
"name": "...",
"unit": "...",
}
```

Note: refer to note for `GET /simulation-unit`

c) **`GET /simulation-unit/default`** ► *find default simulation-unit, which is first simulation-unit stored in database*
Response body on success:

```
{
"id": 576b7471040bc30d1088fdb3,
"name": "...",
"unit": "...",
}
```

Note: refer to note for `GET /simulation-unit`

d) **`POST /simulation-unit`** ► *Submit a new simulation unit request*
Parameters:
- `name` (String): Simulation Unit's name
- `unit` (String): Simulation Unit's content description described in `GET /simulation-unit`

Response body on success: refer to response body for `GET /simulation-unit`

e) **`PUT /simulation-unit/{objectId}`** ► *Update an existing simulation unit specified by objectId*
Parameters:

- `objectId` (String): Simulation Unit's objectId to be updated
- `name` (String): Simulation Unit's name
- `unit` (String): Simulation Unit's content description described in `GET /simulation-unit`

Response body on success: refer to response body for `POST /simulation-unit`

f) **`DELETE /simulation-unit/{objectId}`** ▶ *delete an existing simulation unit specified by objectId*
Response body on success: None

Operations on `simulation`

a) **`GET /simulation`** ▶ *List all simulations*
Response body on success:


```
[
{
"id": 576b7471040bc30d1088fdb3,
"filePath": "...",
"simulationDescription": "...",
"simulationName": "...",
"timeCreated": "...",
"timeFinished": "..."
},
...
]
```


Note:

- `id` is an auto-generated unique object id of the simulation
- `filePath` is the path of the CSV file that is to store simulation result, which is created after simulation finish.
- `timeCreated` is the time when the simulation started.
- `timeFinished` is the time when the simulation finished.

b) **`GET /simulation/file`** ▶ *prepares CSV file for simulation to be downloaded*
Parameters:

- `filePath` (String): The Path of the CSV File of selected simulation

Response body on success:

- `CSV File` as Content-Disposition in Response Header shown in Figur A.1

c) **`POST /simulation`** ▶ *starts a new simulation and saves into database*
Parameters:

- `units` (List<String>): Selected Simulation Units converted to String for simulation algorithm
- `tasks` (List<String>): Selected Simulation Tasks converted to String for simulation algorithm
- `composerProperties` (String): A String converted from JSON Object, stores simulation algorithm properties (JSON File shown in Listing...)

clock	task_id	flag	algo_time	task_name	data	task	solution_components	objective_value	cost	norm_cost
250.17142857142858	1	f	9149695	HumanSensing	null	Task #1 (Queued)	[1 Task #1 (Queued) S#HBS_Collector_78 0.0] [2 Task #1 (Queued) S#HBS_Assessor_46 0.0]	-4.4942328371557893E307	3.4834445690126126	0.9965276514739784
250.34285714285716	2	f	3380137	HumanSensing	null	Task #2 (Queued)	[1 Task #2 (Queued) S#HBS_Collector_58 0.0] [2 Task #2 (Queued) S#HBS_Assessor_95 0.0]	-4.4942328371557893E307	3.779638285747229	0.9962336531864295
250.51428571428573	3	f	4061681	HumanSensing	null	Task #3 (Queued)	[1 Task #3 (Queued) S#HBS_Collector_15 0.0] [2 Task #3 (Queued) S#HBS_Assessor_38 0.0]	-4.4942328371557893E307	3.6827384589205616	0.9963306634107415
250.6857142857143	4	f	3034879	HumanSensing	null	Task #4 (Queued)	[1 Task #4 (Queued) S#HBS_Collector_86 0.0] [2 Task #4 (Queued) S#HBS_Assessor_40 0.0]	-4.4942328371557893E307	2.402544463779195	0.9976055624902996
250.8571428571429	5	f	1712446	HumanSensing	null	Task #5 (Queued)	[1 Task #5 (Queued) S#HBS_Collector_146 0.0] [2 Task #5 (Queued) S#HBS_Assessor_130 0.0]	-4.4942328371557893E307	3.654201819515948	0.9963546044758427
251.02857142857147	6	f	171707	MachineSensing	null	Task #6 (Queued)	[1 Task #6 (Queued) S#HBS_Sensor_243 0.0]	-4.4942328371557893E307	3.1013830112299847	0.9969062241169511
251.20000000000005	7	f	165181	MachineSensing	null	Task #7 (Queued)	[1 Task #7 (Queued) S#HBS_Sensor_204 0.0]	-4.4942328371557893E307	3.442081394837121	0.9965690386710695
251.37142857142862	8	f	141174	MachineSensing	null	Task #8 (Queued)	[1 Task #8 (Queued) S#HBS_Sensor_224 0.0]	-4.4942328371557893E307	2.3690230889414456	0.9976385016000607
251.5428571428572	9	f	139436	MachineSensing	null	Task #9 (Queued)	[1 Task #9 (Queued) S#HBS_Sensor_242 0.0]	-4.4942328371557893E307	2.6788804566519153	0.9973283604070281
251.71428571428578	10	f	138171	MachineSensing	null	Task #10 (Queued)	[1 Task #10 (Queued) S#HBS_Sensor_207 0.0]	-4.4942328371557893E307	3.008487170011768	0.9969957613357107
251.88571428571436	11	f	1137065	HumanSensing	null	Task #11 (Queued)	[1 Task #11 (Queued) S#HBS_Collector_98 0.0] [2 Task #11 (Queued) S#HBS_Assessor_101 0.0]	-4.4942328371557893E307	3.6411756110010147	0.9963783496394288
252.05714285714294	12		926442	HumanSensing	null	Task #12 (Queued)	[1 Task #12 (Queued) S#HBS_Collector_32 0.0] [2 Task #12 (Queued) S#HBS_Assessor_258 0.0]	-4.4942328371557893E307	6.217898029527972	0.9938214252399076
252.2285714285715	13	f	714475	HumanSensing	null	Task #13 (Queued)	[1 Task #13 (Queued) S#HBS_Collector_11 0.0] [2 Task #13 (Queued) S#HBS_Assessor_155 0.0]	-4.4942328371557893E307	3.8127280359838167	0.9962003681397913
252.4000000000001	14		743422	HumanSensing	null	Task #14 (Queued)	[1 Task #14 (Queued) S#HBS_Collector_259 0.0] [2 Task #14 (Queued) S#HBS_Assessor_68 0.0]	-4.4942328371557893E307	5.811911346497818	0.9942115219130757
252.57142857142867	15	f	965635	HumanSensing	null	Task #15 (Queued)	[1 Task #15 (Queued) S#HBS_Collector_145 0.0] [2 Task #15 (Queued) S#HBS_Assessor_35 0.0]	-4.4942328371557893E307	5.0355443671743885	0.9949888686568567
252.74285714285725	16	f	93378	MachineSensing	null	Task #16 (Queued)	[1 Task #16 (Queued) S#HBS_Sensor_245 0.0]	-4.4942328371557893E307	3.654607498648004	0.9963574556655276
252.91428571428582	17	f	113822	MachineSensing	null	Task #17 (Queued)	[1 Task #17 (Queued) S#HBS_Sensor_233 0.0]	-4.4942328371557893E307	3.6728775990750107	0.9963394330764577
253.0857142857144	18	f	96605	MachineSensing	null	Task #18 (Queued)	[1 Task #18 (Queued) S#HBS_Sensor_234 0.0]	-4.4942328371557893E307	2.46187862262727	0.9975414953317401
253.25714285714298	19	f	89887	MachineSensing	null	Task #19 (Queued)	[1 Task #19 (Queued) S#HBS_Sensor_210 0.0]	-4.4942328371557893E307	2.912133062314972	0.9970961696032195
253.42857142857156	20	f	89574	MachineSensing	null	Task #20 (Queued)	[1 Task #20 (Queued) S#HBS_Sensor_241 0.0]	-4.4942328371557893E307	3.713888911552603	0.9962959333908088
253.60000000000014	21	f	637163	HumanSensing	null	Task #21 (Queued)	[1 Task #21 (Queued) S#HBS_Collector_67 0.0] [2 Task #21 (Queued) S#HBS_Assessor_135 0.0]	-4.4942328371557893E307	3.9861454324372825	0.9960344702408541
253.77142857142871	22	f	804267	HumanSensing	null	Task #22 (Queued)	[1 Task #22 (Queued) S#HBS_Collector_195 0.0] [2 Task #22 (Queued) S#HBS_Assessor_87 0.0]	-4.4942328371557893E307	4.541973745726807	0.9954765896752595
253.9428571428573	23	f	1913495	HumanSensing	null	Task #23 (Queued)	[1 Task #23 (Queued) S#HBS_Collector_18 0.0] [2 Task #23 (Queued) S#HBS_Assessor_199 0.0]	-4.4942328371557893E307	4.76688458419075	0.9952498646584104
254.11428571428587	24	f	681814	HumanSensing	null	Task #24 (Queued)	[1 Task #24 (Queued) S#HBS_Collector_175 0.0] [2 Task #24 (Queued) S#HBS_Assessor_71 0.0]	-4.4942328371557893E307	4.186665198902648	0.9958296871076348
254.28571428571445	25	f	3001621	HumanSensing	null	Task #25 (Queued)	[1 Task #25 (Queued) S#HBS_Collector_94 0.0] [2 Task #25 (Queued) S#HBS_Assessor_192 0.0]	-4.4942328371557893E307	3.972494831647719	0.9960486163022632
254.45714285714303	26	f	139307	MachineSensing	null	Task #26 (Queued)	[1 Task #26 (Queued) S#HBS_Sensor_207 0.0]	-4.4942328371557893E307	3.008487170011768	0.9970014722402181
254.6285714285716	27	f	110647	MachineSensing	null	Task #27 (Queued)	[1 Task #27 (Queued) S#HBS_Sensor_220 0.0]	-4.4942328371557893E307	2.521141752636334	0.9974805507662111
254.80000000000018	28	f	287059	MachineSensing	null	Task #28 (Queued)	[1 Task #28 (Queued) S#HBS_Sensor_228 0.0]	-4.4942328371557893E307	3.018654662414993	0.996989623520326
254.97142857142876	29	f	88078	MachineSensing	null	Task #29 (Queued)	[1 Task #29 (Queued) S#HBS_Sensor_201 0.0]	-4.4942328371557893E307	2.628507517263931	0.9973813069407027
255.14285714285734	30	f	151767	MachineSensing	null	Task #30 (Queued)	[1 Task #30 (Queued) S#HBS_Sensor_218 0.0]	-4.4942328371557893E307	2.9476417562696806	0.9970601533512179
255.31428571428592	31	f	824971	HumanSensing	null	Task #31 (Queued)	[1 Task #31 (Queued) S#HBS_Collector_112 0.0] [2 Task #31 (Queued) S#HBS_Assessor_42 0.0]	-4.4942328371557893E307	4.996143867337386	0.995033987341398

Figure A.1: Simulation Result as CSV File

- `consumerProperties` (Object): An Object stores following information :
  * `numberOfCycles` (int):
  * `waitBetweenCycles` (int): delay (in simulation time unit) between each cycle
  * `tracerConfig` (String): the path of the tracer.json file
- `simulation` (Object): An Object stores following simulation information :
  * `id` (String, optional): the unique objectId of simulation
  * `timeCreated` (String): the time when the simulation started
  * `timeFinished` (String, optional): the time when the simulation ended.
  * `simulationName` (String): name of the simulation given by end user
  * `simulationDescription` (String): stores a detailed information about simulation
  * `filePath` (String, optional): the path of the CSV file of simulation result

Response body on success: None

d) **`POST /simulation/graph`** ▶ *Submit a graph chart image request*

Parameters:

- `xAxis` (String): first metric element of metric correlation analyzer
- `yAxis` (String): second metric element of metric correlation analyzer
- `simulationId` (string): the object id of simulation which will be searched

Response body on success:

- `id` (String): the unique object id of graph

- image (String): a String representation of graph
- timeCreated (string): the time when graph is created

Glossary

compute unit is a resource providing services capable of processing input data into a more useful information in a (semi-)automated manner.

Bibliography

[1] Muhammad Zuhri Catur Candra. *Hybrid Human-Machine Computing Systems.* PhD thesis, Technische Universität Wien, 2016.