BOTANICAL GARDEN CONTROL SYSTEM

Second Deliverable of Software Quality Engineering

Agostino Mascitti (matriculation number: 250225)
Stephane Vanberg (matriculation number: 247057)
Yuna Phrolov (matriculation number: 247049)

January 2017

Contents

1 Introduction

The second homework consists in deriving a performance model from the UML model of the first homework. One first obtains a software model using execution graphs. Queuing network are then used to represent the hardware platform. And finally, the last section depicts the AEmilia model. In addition to this report, one provides:

- MagicDraw reworked UML model: `UML/reworkedBGCS.mdzip`
- JMT models:
  - Initial model: `JMT/QN.jsimg`
  - Refactored model: `JMT/refactoredQN.jsimg`
- Detailed computations of the demand vectors: `Demand Vectors.xlsx`
- AEmilia:
  - Initial model: `Aemilia/BGCS/BGCS.aem` and `Aemilia/BGCS/BGCS.rew`
  - Refactorization 1: `Aemilia/BGCS_2/BGCS.aem` and `Aemilia/BGCS_2/BGCS.rew`
  - Refactorization 2: `Aemilia/BGCS_3/BGCS.aem` and `Aemilia/BGCS_3/BGCS.rew`
  - Refactorization 4: `Aemilia/BGCS_4/BGCS.aem` and `Aemilia/BGCS_4/BGCS.rew`

2 Annotated Deployment Diagram

We have made the following modifications to the Deployment Diagram:

- Added the artifacts *Application*, *Camera*, *Humidity Sensor*, *Gates*, *Greenhouse Management*, *Guide*, *Garden State Handling*, *Database*
- Removed the artifact *FileOfDB*
- Remove the *Web Server* execution environment
- Separated the device *Database* from the *Server*
- Removed the *Moisture Sensor* that was not used
- Added a connection between *Mobile* and *Camera* devices

The modifications mainly consist in deploying the software components on hardware nodes. The resulting deployment diagram is shown on figure 9. As we can see, the the deployment diagram was also annotated with *CPUs* and *Disks*.

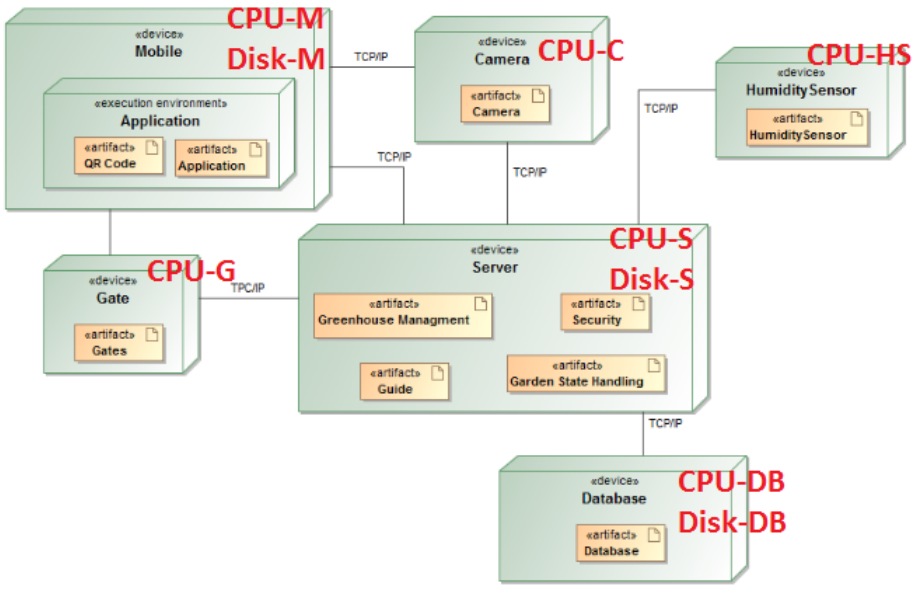

Figure 1: Annotated Deployment Diagram

3 Performance Requirements

Before specifying the performance requirements, one assumes a workload of 5 administrators and visitors entering a greenhouse every 10 seconds. The two performance requirements are:

- **Response Time:** the average response time of the administrator mobile application must be less than 1 seconds.
- **Utilization:** Each device should have, in average, an utilization below 70%.

4 Execution Graphs

The goal of this section is to derive a performance model from the UML model provided in the first homework. Two use cases (and therefore two sequence diagrams) are modeled into an execution graphs, which are then parametrized. As a result one obtains the demand vectors for each of the two classes of jobs: *visitors* (UC1) and *administrators* (UC2). This software model is then mapped to hardware resources from which mean service times are computed.

4.1 Software Resources

The basic nodes of the execution graphs are parametrized with software demand vectors (number of visits). The *software resources* are the following:

- **WorkUnit:** number of computational operations
- **DB:** number of database requests
- **Net:** number of messages sent (or received) over the network
- **Sensing:** number sense of a sensor

The virtual demand vectors are thus vectors of length 4. For example, a basic node requesting 2 Work units and 1 Network message will be parametrized with:

$$d = [2, 0, 1, 0]$$

4.2 Enter Greenhouse

The execution graph corresponding to this use case is shown on figure 2. Notice that the two branches of the conditional nodes have been annotated with probabilities.

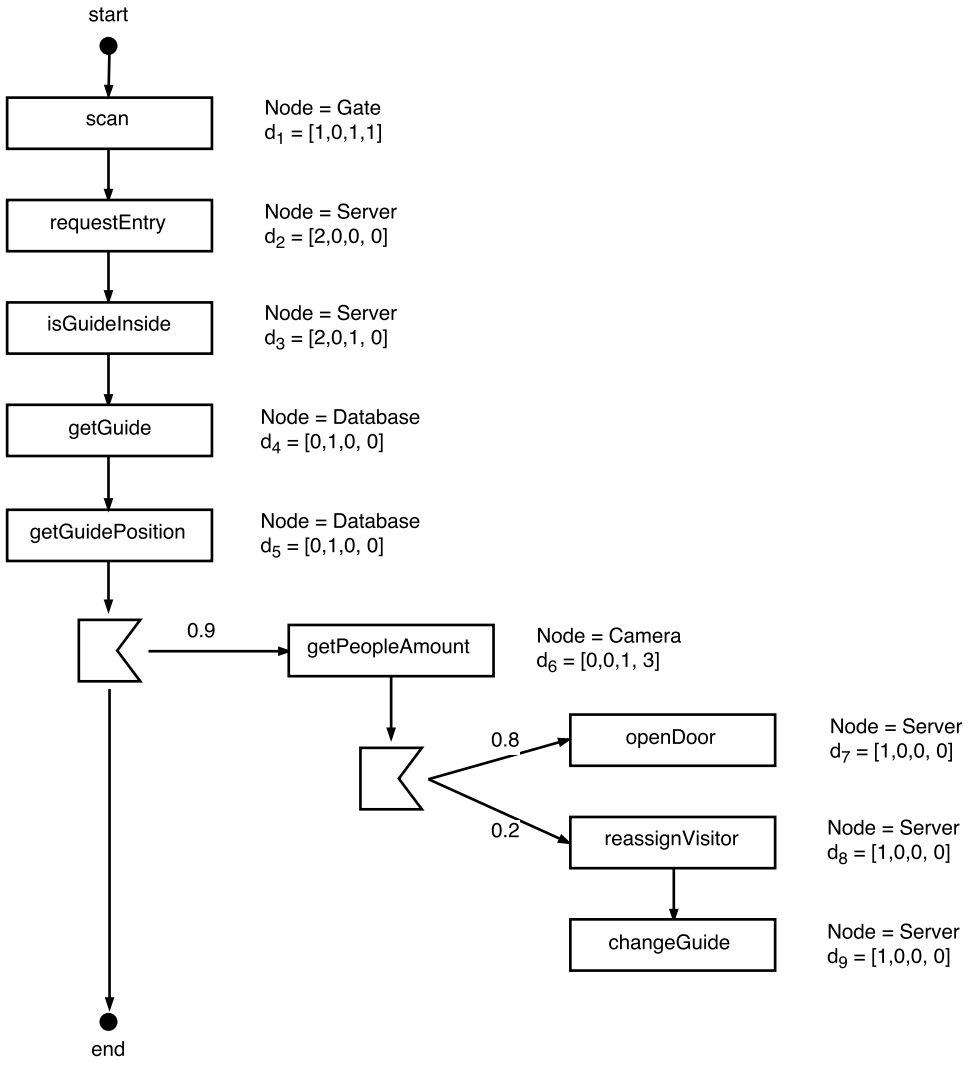

Figure 2: Execution graph 1

From this annotated execution graph, one can compute the demand vectors associated to each deployment node by applying the synthesis rules.

$$
\begin{aligned}
d_{EG1,Gate} &= d_1 \\
&= \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} \\
d_{EG1,Server} &= d_2 + d_3 + 0.9 \cdot 0.8 \cdot d_7 + 0.9 \cdot 0.2 \cdot (d_8 + d_9) \\
&= \begin{bmatrix} 5.17 & 0 & 1 & 0 \end{bmatrix} \\
d_{EG1,Camera} &= 0.9 \cdot d_6 \\
&= \begin{bmatrix} 0 & 0 & 0.9 & 2.7 \end{bmatrix} \\
d_{EG1,Database} &= d_4 + d_5 \\
&= \begin{bmatrix} 0 & 2 & 0 & 0 \end{bmatrix}
\end{aligned}
$$

4.3 Greenhouse Change Settings

One now applies the same method to derive the remaining parametrized EGs. The second execution graph is provided on figure 3.

$$
\begin{aligned}
d_{EG2,Mobile} &= d_1 + 0.05 \cdot d_5 + 0.95 \cdot (d_6 + d_7 + 0.3 \cdot d_{10} + 0.3 \cdot d_{12} + 0.1 \cdot d_{14} + 0.25 \cdot d_{17} + 0.25 \cdot d_{20}) \\
&= \begin{bmatrix} 7.13 & 0 & 3.09 & 0 \end{bmatrix} \\
d_{EG2,Server} &= d_2 + d_4 + 0.95 \cdot (d_8 + 0.3 \cdot d_{11} + 0.3 \cdot d_{13} + 0.1 \cdot d_{15} + 0.25 \cdot d_{18} + 0.25 \cdot d_{21}) \\
&= \begin{bmatrix} 7.04 & 0 & 1.95 & 0 \end{bmatrix} \\
d_{EG2,Database} &= d_3 + 0.95 \cdot d_9 \\
&= \begin{bmatrix} 1.95 & 5.85 & 1 & 0 \end{bmatrix} \\
d_{EG2,Camera} &= 0.95 \cdot (0.5 \cdot d_{16} + 0.5 \cdot d_{19}) \\
&= \begin{bmatrix} 2.85 & 0 & 0.95 & 0 \end{bmatrix}
\end{aligned}
$$

4.4 Overhead matrix

This step consists in obtaining the real demand vectors by means of an *overhead matrix*. The virtual resources will be mapped to the physical resources of each deployment node, at a lower level of abstraction. Since we have 4 virtual resources and 3 different devices (CPU, Disk, Network), the overhead matrix is a 4x3 matrix[1].

Device	CPU	Disk	Network
Service Unit	K-instr.	Physical I/O	Message

Table 1: Hardware Service Units

$$
M = \begin{bmatrix} 1200 & 200 & 0 \\ 200 & 800 & 0 \\ 100 & 100 & 500 \\ 400 & 0 & 0 \end{bmatrix}
$$

where $[M]_{i,j}$ = amount of physical resources j required by a unit of virtual resource i. For example, one WorkUnit corresponds to 12 K-CPU instructions. The demand vectors are finally obtained by multiplying the virtual demand vectors by the matrix M and using the knowledge of the deployment node on which the computation is performed.

[1]The computations are performed here in a distributed settings, but we use the same matrix for all devices. Only the service time will differ.

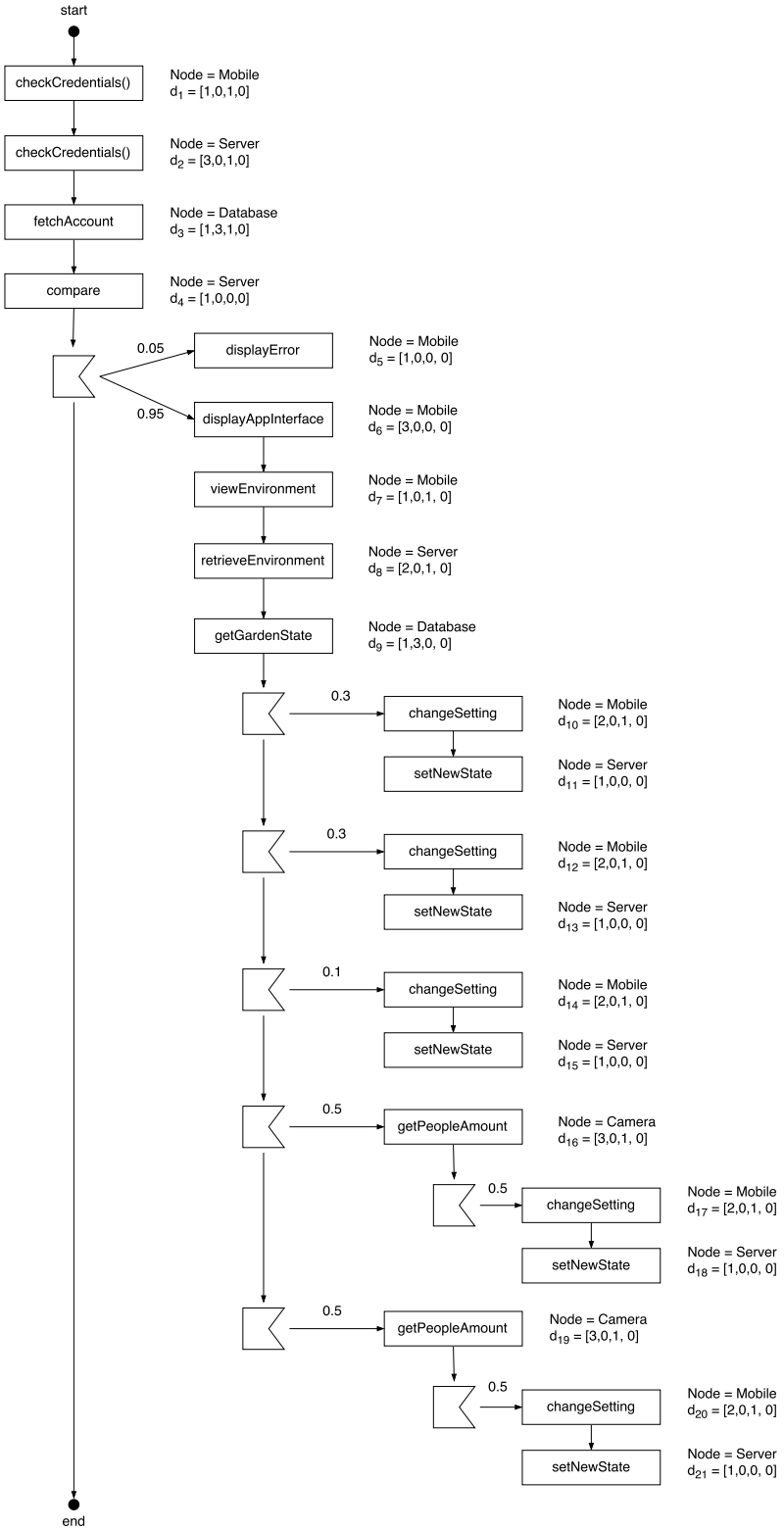

Figure 3: Execution graph 2

The results are summarized in the table 2.

	CPU-M	**Disk-M**	**CPU-S**	**Disk-S**	**CPU-DB**	**Disk-DB**	**CPU-C**	**CPU-G**
Visitors	0	0	6304	1134	400	1600	400	1700
Administrators	8865	1735	8643	1603	3515	665	3610	0
Service time (sec)	0.0001	0.0004	0.00008	0.0008	0.0003	0.0006	0.0004	0.0007
Visitor Service time	0	0	0.50432	0.9072	0.12	0.96	0.16	1.19
Admin Service time	0.8865	0.694	0.69144	1.2824	1.0545	0.399	1.444	0

Table 2: Overhead Matrix processing

As we can see, the sensors are not used. It was expected since, in the second sequence diagram, the data are supposed to be already present in the database and not sensed in real-time by the sensors. Network delays are assumed to be negligible in our system (same LAN). But the fact of sending messages required some CPU resources. This is why the network virtual resource is still defined.

5 Queuing Network

This sections describes the hardware performance model. Firstly, the queueing network topology is derived from the deployment diagram. Then it is parametrized with the results obtained in table 2. More precisely, the two last rows represent the mean service time of each class of jobs. In other words the parameter $1/\lambda$ is set to this value in the corresponding service center in JMT.

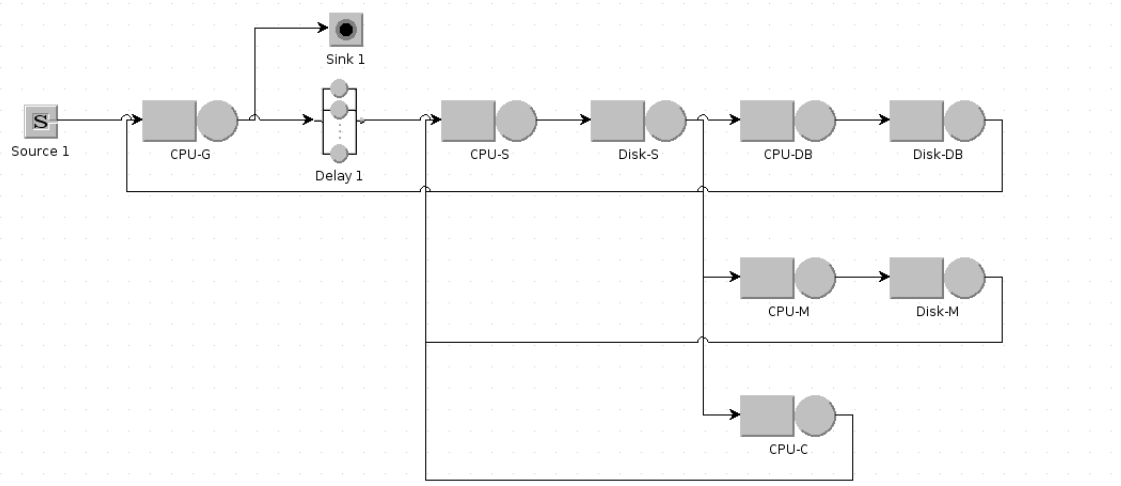

Figure 4: Queuing network topology

5.1 Class of jobs

The two class of jobs obtained from the two use cases are introduced in the JMT model.

- **Visitors**: as an open class. Visitors are assumed to arrive every 10 seconds
- **Administrator**: as a closed class. A fixed number of $N = 5$ administrators is present in the system. They all have a think time of $Z = 10$ seconds.

5.2 Simulation results

The performance indexes corresponding to the performance requirements are shown in the following figures.

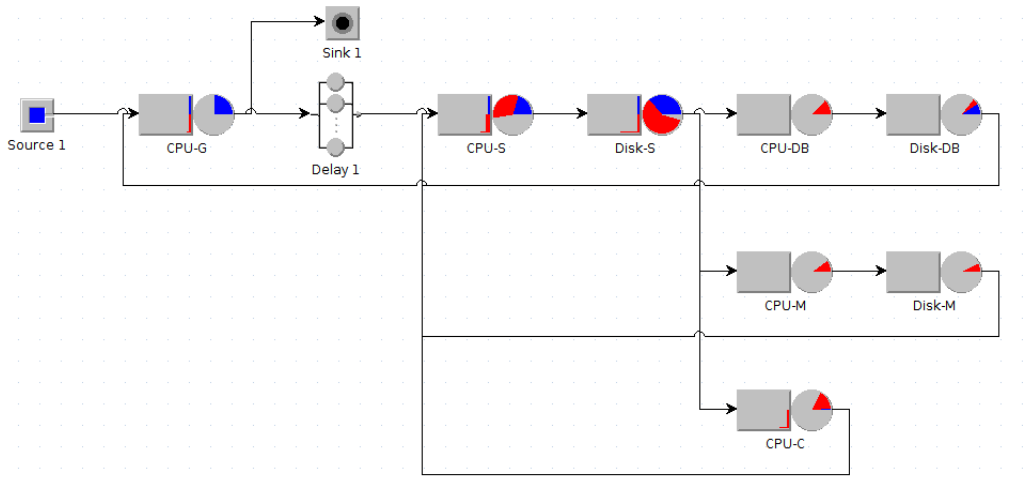

Figure 5: Queue and utilization

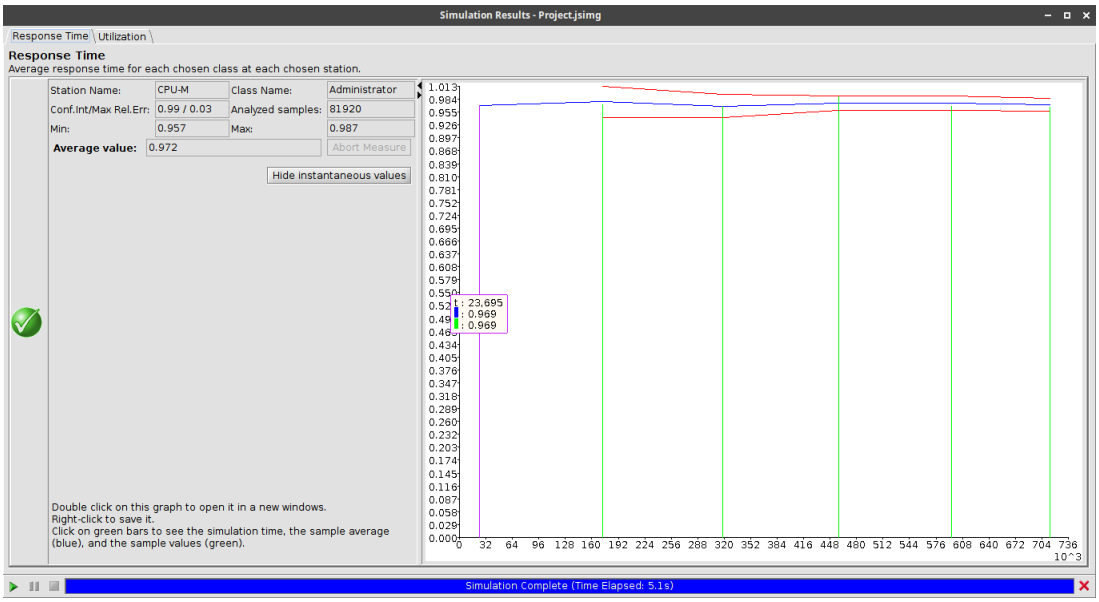

Figure 6: Response Time

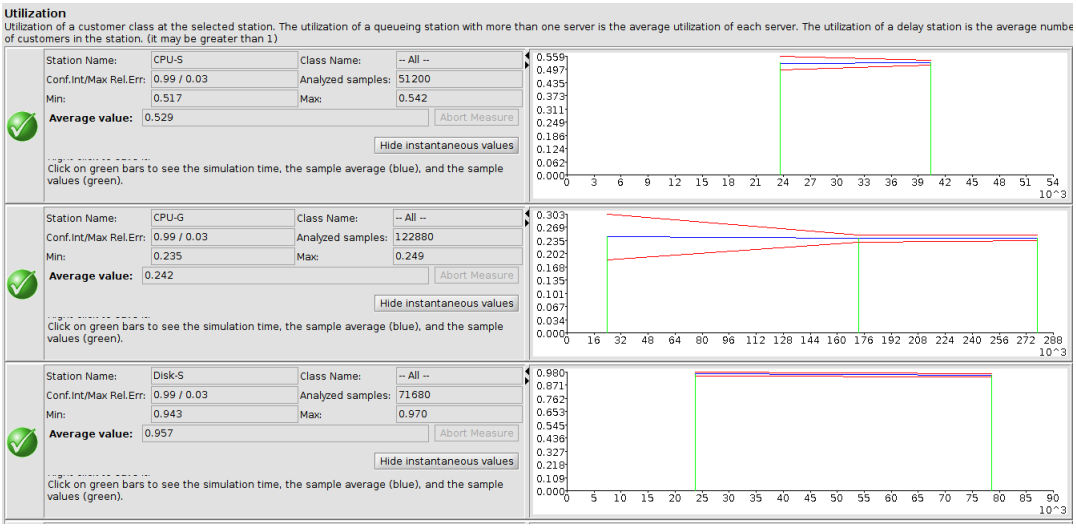

Figure 7: Utilization (part 1)

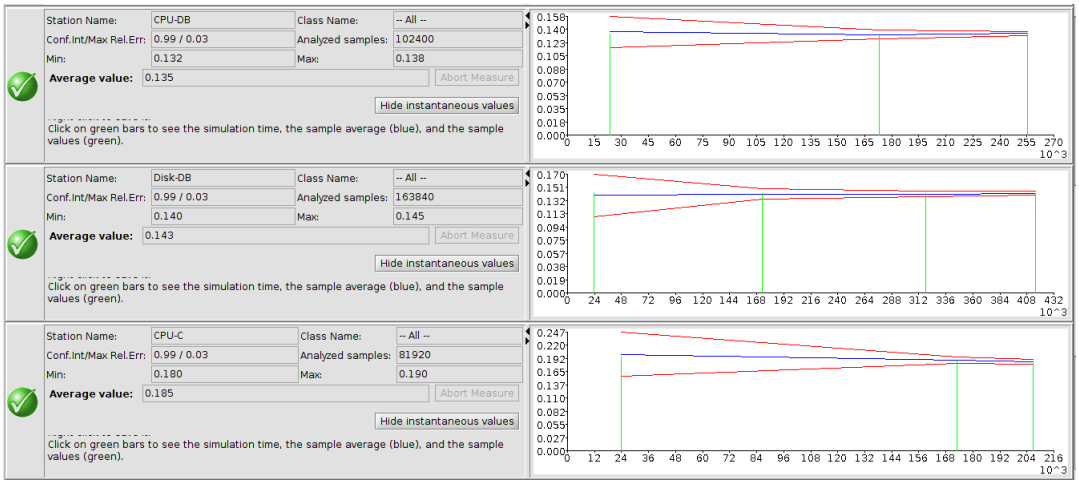

Figure 8: Utilization (part 2)

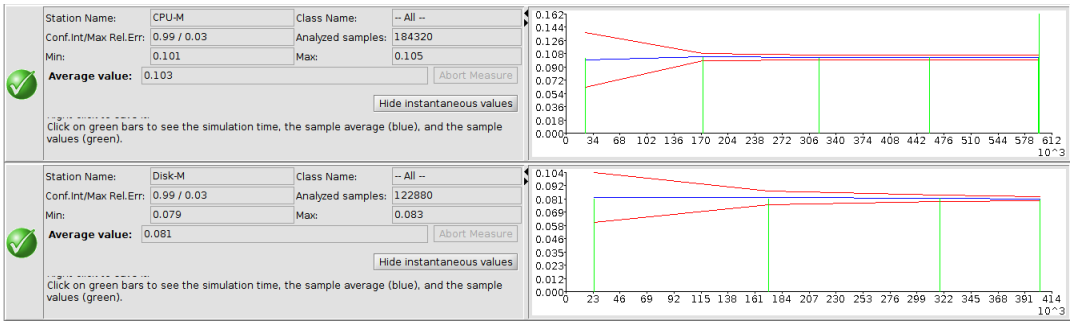

Figure 9: Utilization (part 3)

5.3 Bottleneck identification

The response time requirement is satisfied by the the system but this is not the case for the utilization one. As we can seen, the server disk (Disk-S) has reaches an average utilization of 95.7%. This is the bottleneck of the system. To address this problem, a second disk of same capacity is added in parallel.

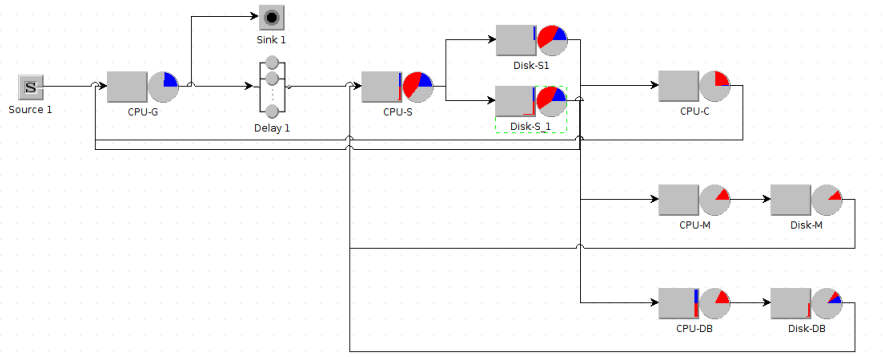

Figure 10: Refactored QN

The refactored queuing network is provided in figure 10. With this new model, on obtains all utilization below 70%. The Response Time requirement is still satisfied.

6 AEmilia

6.1 Modelling

The AEmilia model can be found in the file `BGCS.eam`

6.1.1 Structure Overview (static specification)

Let's first present an overview of the model. The AEmilia model is shown in figure 11. As we can see, our model is composed of six Architecture Element Types (AET):

- **App_Type:** represents the application an administrator of the BGCS works with. The application is used to monitor a greenhouse and potentially change settings (activate heater, fan, lamp etc.).
- **Net_Type:** is a symmetric bi-directional network used to introduce a delay between the application and the service providers. A network delay is introduced only between the application and the remaining components because it is assumed to be negligible within the remaining components. Indeed, the administrator may be using the application from several locations while the sensors, for instance, should be close to the database.
- **Sec_Type:** is the security component responsible for authenticating users by fetching accounts from a database
- **GH_Type:** is the greenhouse component returning the status of the greenhouse, from a database regularly populated by sensors data. This component is also used to change some settings inside the greenhouse.
- **DB_Type:** models the centralized database containing the administrators accounts and the sensors data.

- **Sensor_Type:** consists in a sensor such as a humidity sensor or a moisture sensor. A sensor frequently reports sensing data in the database. Many instances of this AET are deployed.

All the Architecture Element Instances presented here are connected by attachments. No Architectural Interactions have been used. More precisely, except for the data updates sent by the sensors to the database which is a or-interaction, the other AEI communicate point-to-point by uni-interactions.

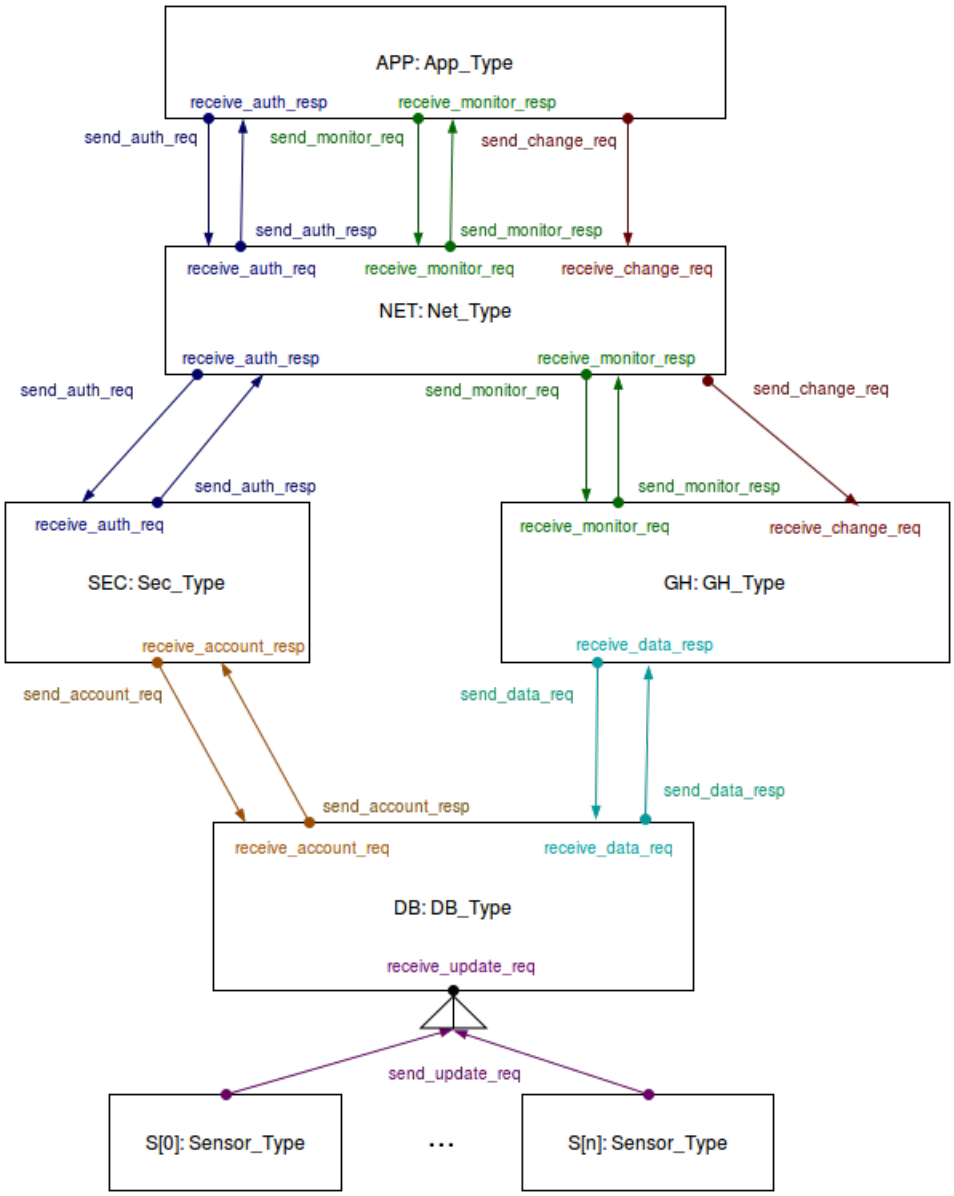

Figure 11: Flowgraph

6.1.2 Behavior (dynamic specification)

We will now detail the behavior of each AET thanks to a state diagram. The behavior of each component is specified in a process algebra from which a state diagram is derived.

Application The administrator has first to authenticate, then he can monitor the status of a greenhouse. And potentially, the status of the greenhouse may be modified with a further request. Since it is not always required, a probability allows this additional request to be performed from time to time.

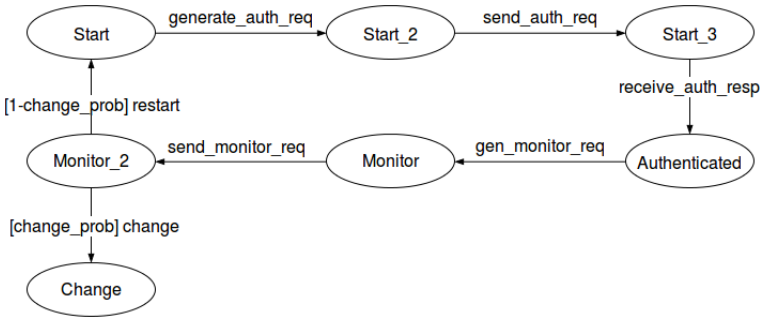

Figure 12: Application State Diagram

Network The network simply forwards the incoming requests. One expect the five types of requests to be forwarded as shown on the state chart.

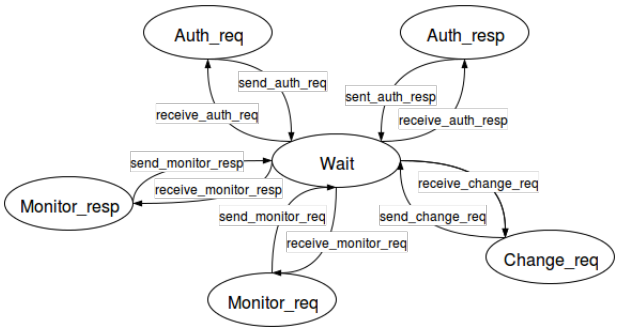

Figure 13: Network State Diagram

Security The security component behave like a buffer receiving authentication requests. Once there are requests to handle, the component fetch the account in the database to check the user credentials. And finally the response is sent back.

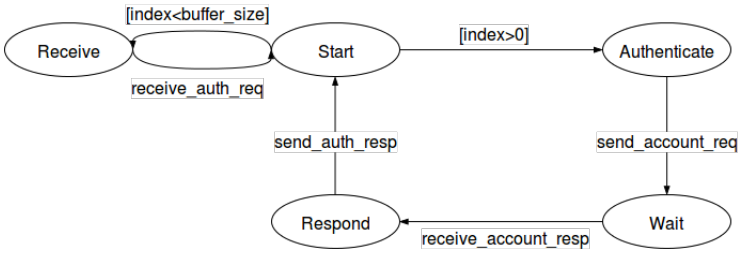

Figure 14: Security State Diagram

Greenhouse The greenhouse reacts to two incoming requests. The first one, `monitor_req`, is handled by requesting data from the database before replying (similarly to the security component). The second request, `change_req`, is directly handled by this component.

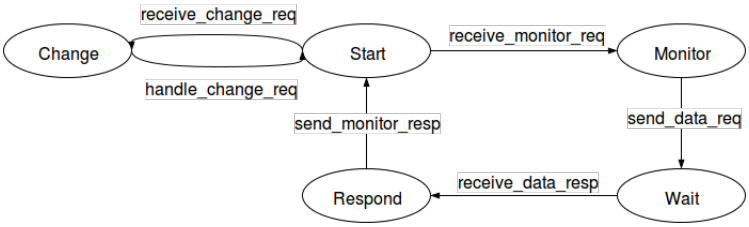

Figure 15: Greenhouse State Diagram

Database The database is used to fetch either accounts or sensor data. From the Start state, depending on the type of request, it goes in one of the three possible state: Account, Data or Update. For the first two states, the data are fetched and then returned while in the third state, the update is handled but no data is sent back.

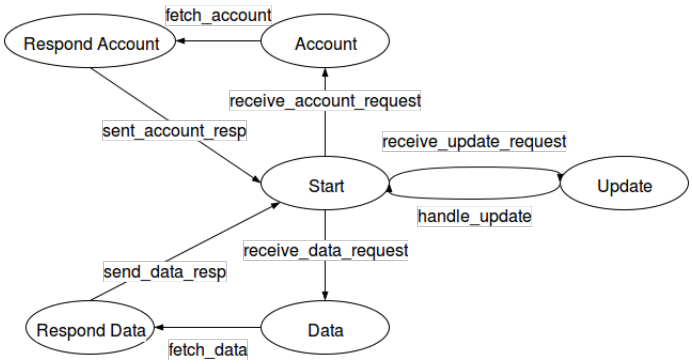

Figure 16: Database State Diagram

Sensors The sensors have a very simple behavior. A timer activate them. They then enter in the sense state, from which they leave after sending the updated data to the database.

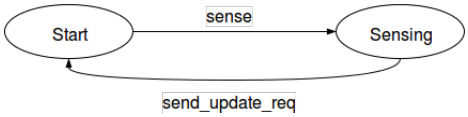

Figure 17: Sensors State Diagram

To conclude this section, notice that the AET accept their service rate as a parameter. The change probability can also be specified as well as the buffer and the number of sensors.

6.2 Analysis

The performance index are computed thanks to the file `BGCS.rew`.

6.2.1 Results

One measures the throughput and the utilization for the different components (except the application). The model presented so far gives the following results:

	BGCS	BGCS 2	BGCS 3	BGCS 4
NET_utilization	0.149816	0.245742	0.295155	0.301307
NET_throughput	224.724	368.613	442.733	451.96
SEC_utilization	0.159379	0.261428	0.313995	0.320539
SEC_throughput	47.8137	78.4284	94.1986	96.1617
GH_utilization	0.129097	0.211757	0.254336	0.259637
GH_throughput	95.6274	156.857	188.397	192.323
DB_utilization	0.88988	0.686366	0.51509	0.50498
DB_throughput	47.8137	78.4284	94.1986	96.1617

Table 3: AEmilia Performance Results

As we can see, the database utilization is very high compared to the other components utilization. Therefore one decided to improve the model in three ways:

- **Users:** reduce the number of sensors (BGCS 2)
- **Hardware:** improve the database service rate (BGCS 3)
- **Software:** use a cache to avoid involving the database (BGCS 4)

6.2.2 Refactoring

Limit the number of sensors In the second model, the number of sensors is 3 instead of 5. As a result, the database utilization drops. Indeed there are now less updates from the sensors (note that the update frequency remain the same here, but this is another possible improvement). The throughput and utilization of the other components also increased. The database is the bottleneck of the system.

Database hardware improvement The second improvement consists in improving the database hardware which results in a higher service rate. Improving the `db_rate` variable from 2000 to 3000 reduces the database utilization of 20%. Even if the database utilization drops from the same amount than with the first improvement, this time the utilization and throughput of the other component only slightly increase.

Cache software improvement In this case, one needs to modify the model to introduce a cache in the Security component. Before fetching the accounts from the database, it will first look into the cache to check if the data are present. If possible, the data of the cache are used instead of soliciting the database. The probability to find the data in the cache is set via the variable `cache_prob`, which is set to 0.3 by default (data locality). The impact of this modification is very low, but there is still a slight improvement.