Technical Report

Model of the Air Conditioning (AC) system

Nathalie Moreno · David Bandera · Manuel F. Bertoa · Carlos Canal · Alejandro Pérez-Vereda · Paula Muñoz · Antonio Vallecillo

July 2020

Abstract This example models a smart Air Conditioning (AC) system, which automatically calculates the optimal temperature of a room according to the comfort preferences of the people in the room. This is accomplished in a transparent manner to the users, being their Digital avatars which know the ranges of temperatures where their users feel comfortable, and communicate directly with the AC system when they detect that it supports this interface. We will model here different versions of the system. The first one (called "crisp") does not consider any type of uncertainty, i.e., all numbers are exact figures and all measures are precise, accurate and trustworthy. All sensors, devices and communication networks are reliable too. Then, we start adding different types of uncertainty to the model in the following sections, namely measurement uncertainty (Sect. 2) and occurrence uncertainty (Sect. 3).

Contents

All authors are with the ITIS Software, Universidad de Málaga, Spain. E-mail: {moreno,bandera,bertoa,canal,apvereda,paulam,av}@lcc.uma.es

1 Crisp version

1.1 Structure of the system

A model for the Air Conditioning (AC) system is shown below. Its main elements are the following:

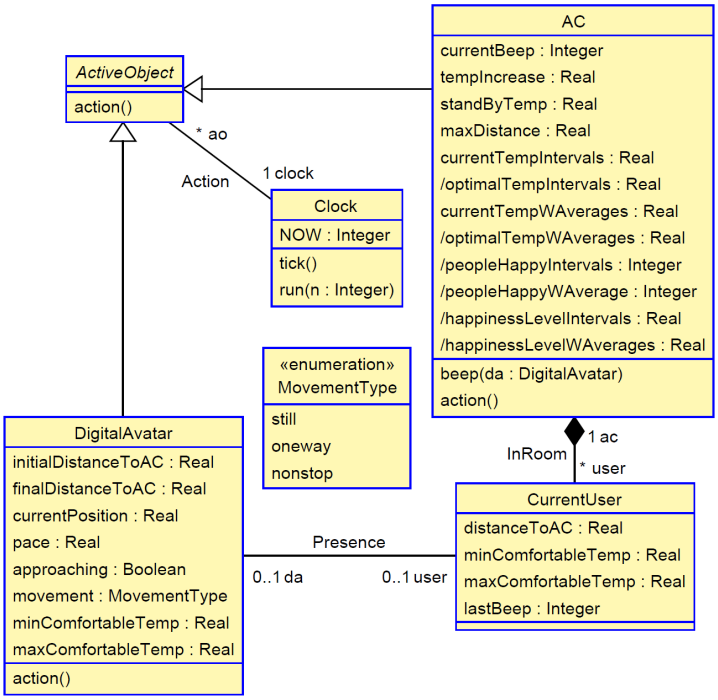


Fig. 1 Air Conditioning Model

- Class `Clock` simulates the passage of time. Its attribute NOW stores the current time (in, e.g., POSIX seconds), and it is incremented every time operation `tick()` is invoked. Operation `run(n:Integer)` invokes n times operation `tick()`.
- Class `DigitalAvatar` represents the digital avatars of users, i.e., the applications that run on the users' mobile phones. Its attributes capture the current distance to the AC controller (`currentPosition`, which is calculated by the phone, depending on the intensity of the Bluetooth signal), as well as the user preferences about the min and max temperature of the room that makes them feel comfortable. Attribute movement is used to distinguish between users who remain still, i.e., sitting in the room, from others moving around. Enumeration `MovementType` defines the types of movements. Literal still is used to represent static users, who do not move

throughout the simulation (and then attributes finaldistancetoAC and pace are ignored). Moving objects move from their initial distance to the AC to their final distance, at a pace given by attribute pace. Once they reach the final position, oneway objects remain there, while nonstop objects reverse their direction and move back to their initial position, continuing forever moving up and down.

- Class `AC` represents the AC controller. Its attributes store the current beep (that coincides with the clock's time), the increase in temperature that is able to carry out every time unit, and the standby temperature. Attribute maxDistance indicates the reach of the AC system, beyond that distance users are not visible and the effects of the AC system are lost. We want to simulate two algorithms for deciding the optimal temperature of the room, depending on the preferences of the users currently in the room, which are represented by instances of class CurrentUser. The first algorithm (Intervals) computes the minimum value that makes more users happy (because it is inside in their comfort ranges). The second one (WAverage) calculates a weighed average using as values the averages of the comfort ranges of each user, and the weights are calculated in inverse proportion to the square of the distance to the AC controller. Other AC attributes indicate how many users are happy with the result of each algorithm, and their average level of happiness (i.e. the average distances between their optimal comfort temperature and that of the AC. Operation `beep()` is invoked by the digital avatars of the users to indicate that they are in the room. The OCL specification of these algorithms is included in Annex 1 of this document.
- Class `CurrentUser` is used to store the information of the users that the AC controller thinks are in the room. They reference the DigitalAvatar instances that they represent and store a copy of the values of their attributes, in case their connection is lost. Attribute lastBeep keeps track of the last time the digital avatar of the user invoked operation `beep()` on the `AC` class.
- Every time a digital avatar invokes a `beep()` operation, if there is no record of the user, the AC controller creates an instance of class `CurrentUser` and links it to the avatar; otherwise, it simply updates the user information with the one provided in the `beep()` operation. In case a digital avatar does not beep() for more than five time units, the AC controller assumes that the corresponding user has left the room, and the associated `CurrentUser` is deleted from the list of users maintained by the AC controller.
- Abstract class `ActiveObject` declares operation `action()`, used to simulate the system execution. On every invocation of the clock's `tick()`, operation `action()` is invoked on every object associated with the clock. Thus every object has the opportunity to carry out the appropriate actions specified in its behavior, emulating a round-robin scheduled system.

The following object diagram shows the state of such a system at one moment in time (NOW=30).

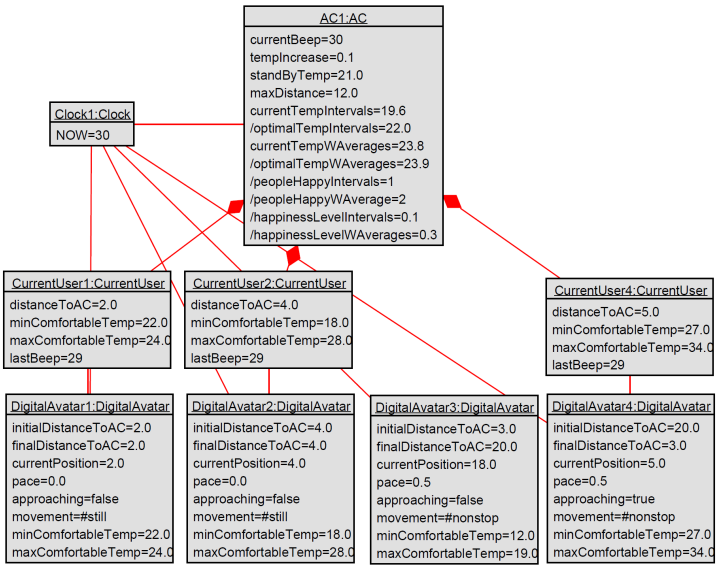


Fig. 2 Object Diagram, now=30

1.2 Adding simulation parameters

To be able to carry out different simulations, we have extended the initial model with a new class (Test) in charge of creating the instances of the simulation with different parameters. The system with this new class is as follows:

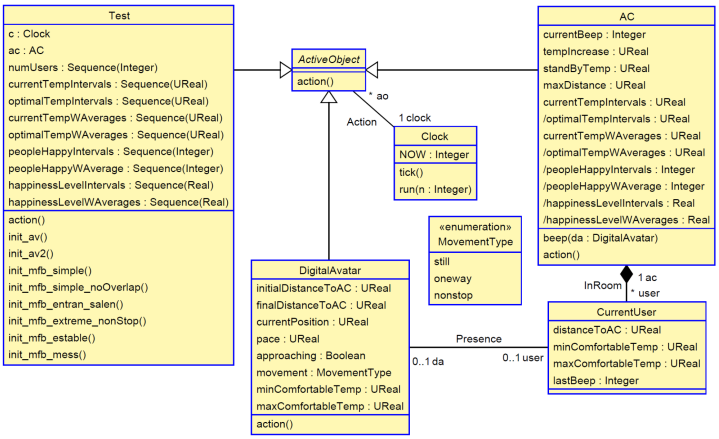


Fig. 3 Simulation Parameters

Class `Test` provides several methods, starting with `init_()`, whose goal is to create the initial system and to set the simulation parameters. Each one

corresponds to a different scenario, as described below. These methods create instances of the `Clock` and `AC` classes (stored in its two first attributes), and the number of DigitalAvatars defined in each scenario. The rest of the attributes of the class Test allow storing the traces of the execution.

We can then simulate the system by simply creating an instance of class Test, initializing the system, and asking the clock to advance. This can be achieved by executing the following USE commands:

```
AC.soil> !new Test('t')
AC.soil> !t.init_av()
AC.soil> !Clock1.run(100)
```

The results can be copied into a csv file and visualized using Excel charts, as we shall see later in Section 5.

1.3 Scenarios

This section describes eight scenarios that we have defined in order to simulate the system and understand how it works in different situations, using both temperature-regulation algorithms.

Each scenario is defined by a name, a set of users, their positions, and the parameters defined in their digital avatars. In the tables, reliable connections are only considered when dealing with Occurrence uncertainty, but have been added here for completeness. The results of the simulation of each scenario are shown, using three charts: (1) the evolution of the room temperatures following the two algorithms, (2) the number of users in the room, and how many of them are happy with the optimal temperatures (when each temperature is within their comfort range), and (3) the average level of happiness of the users in the room (computed as the average difference between the optimal temperature as set by the AC and the average of their comfort range).

1.3.1 Scenario 1 ("av")

Five users, all inside the room at all times. 2 still users, 1 approaching the AC, 1 leaving it, one moving up and down.

AC controller Parameters:

- StandByTemp: 21.0 (temperatura en reposo)
- MaxDistance: 12.0 (podemos suponer que esto también representa el tamaño de la sala)
- tempIncr: 0.1 (cuanto es capaz de cambiar la temperatura en 1 unidad de tiempo)

Users Parameters:

Table 1 Scenario 1.

	InitialDistance ToAC	**finalDistance ToAC**	**Pace**	**Movement**	**minComfort Temp**	**maxComfort Temp**	**Reliable Connection**
User1	3.5	3.5	0.0	still	19.0	21.9	0.99
User2	12.0	3.0	0.25	oneway	22.0	25.0	0.99
User3	4.0	12.0	0.25	oneway	24.0	27.0	0.99
User4	10.0	10.0	0.0	still	20.0	24.0	0.33
User5	12.0	4.0	0.25	nonstop	22.0	26.0	0.99

Results for scenario 1.3.1, CRISP VERSION

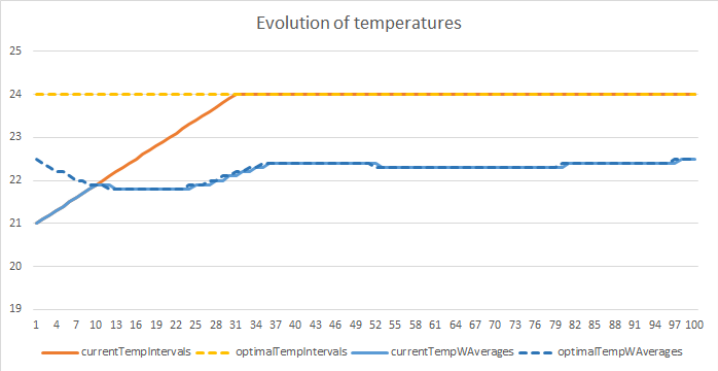
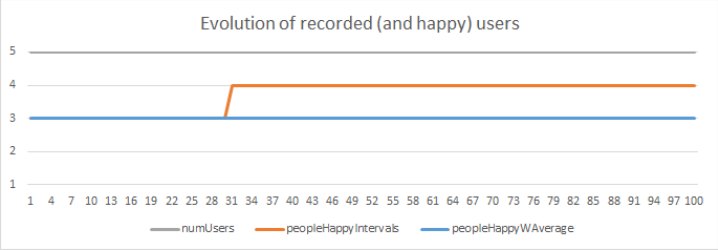
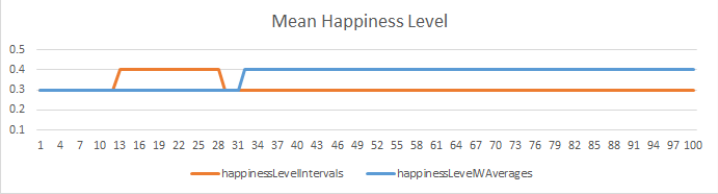


Fig. 4 Scenario 1. Evolution Temperatures



(a)
h!

Fig. 5 Scenario 1. Evolution Recorded Users



(a)
h!

Fig. 6 Scenario 1. Mean Happiness Level

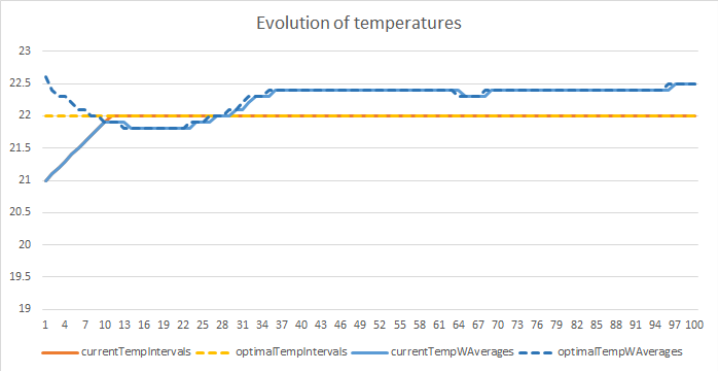
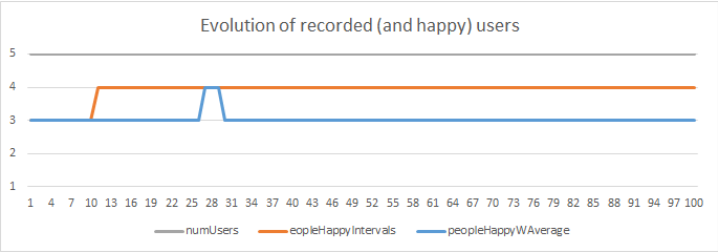
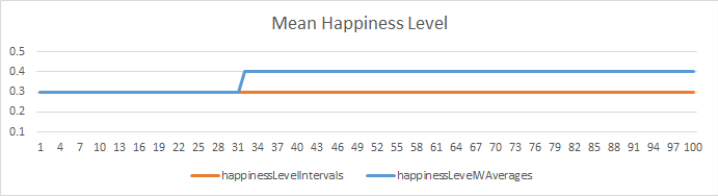


Fig. 7 Scenario 1. Evolution Temperatures 2



(a)
h!

Fig. 8 Scenario 1. Evolution Recorded Users 2



(a)
h!

Fig. 9 Scenario 1. Mean Happiness Level 2

1.3.2 Scenario 2 ("av2")

Five users, 3 of them inside the room at all times. 2 still users, 1 approaching the AC starting from outside the room, 1 leaving the room, one moving up and down, continuously entering and leaving the room.

AC controller Parameters:

- StandByTemp: 21.0 (temperatura en reposo)
- MaxDistance: 12.0 (podemos suponer que esto también representa el tamaño de la sala)
- tempIncr: 0.1 (cuanto es capaz de cambiar la temperatura en 1 unidad de tiempo)

Users Parameters:

Table 2 Scenario 2.

	InitialDistance ToAC	finalDistance ToAC	Pace	Movement	minComfort Temp	maxComfort Temp	Reliable Connection
User1	3.5	3.5	0.0	still	19.0	21.9	0.99
User2	13.0	3.0	0.25	oneway	22.0	25.0	0.99
User3	4.0	15.0	0.25	oneway	24.0	27.0	0.99
User4	10.0	10.0	0.0	still	20.0	24.0	0.33
User5	15.0	8.0	0.25	nonstop	22.0	26.0	0.99

Results for scenario 1.3.2, CRISP VERSION

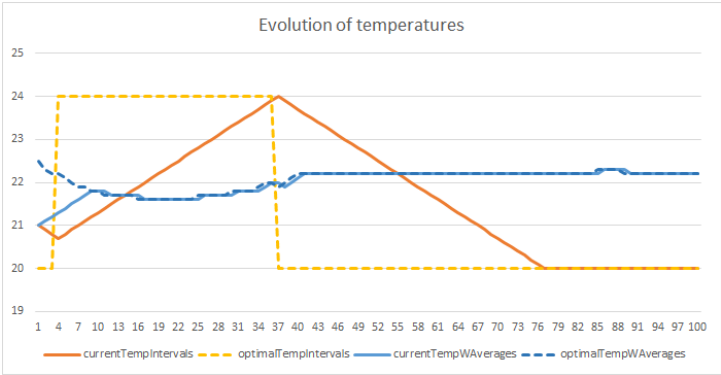


Fig. 10 Scenario 2. Evolution of Temperatures

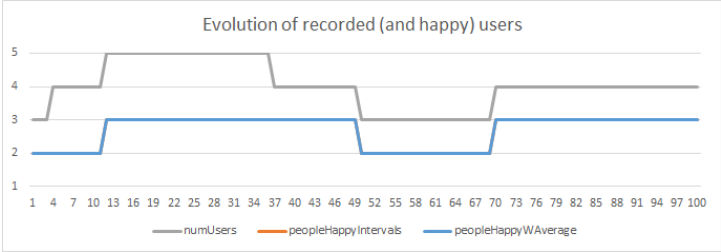


Fig. 11 Scenario 2. Evolution of Recorded Users

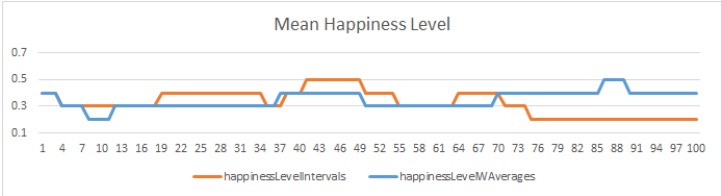


Fig. 12 Scenario 2. Mean Happiness Level

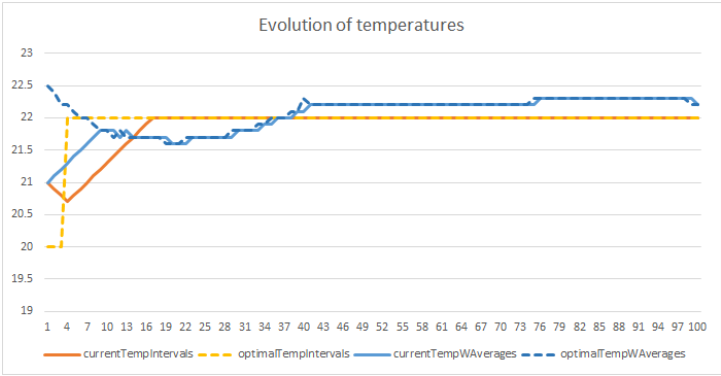


Fig. 13 Scenario 2. Evolution Temperatures 2

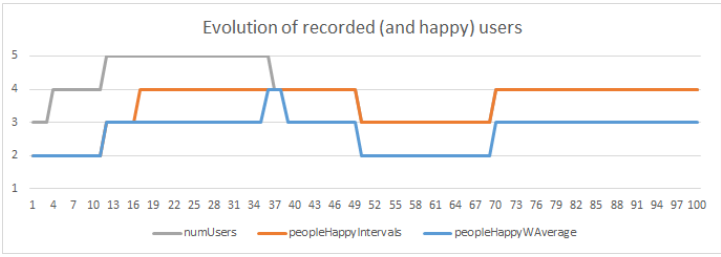


Fig. 14 Scenario 2. Evolution Recorded Users 2

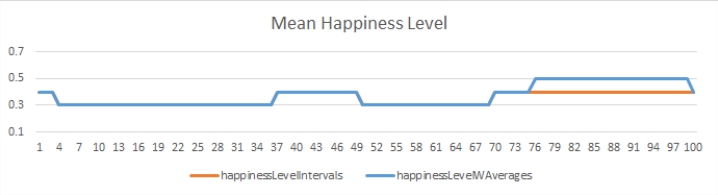


Fig. 15 Scenario 2. Mean Happiness Level 2

1.3.3 Scenario 3 ("MFB-Simple")

Scenario where 3 users arrive gradually with moderate values that matching at 23.0 ºC. They all stay inside the room.

AC controller Parameters:

- StandByTemp: 21.0 (temperatura en reposo)
- MaxDistance: 12.0 (podemos suponer que esto también representa el tamaño de la sala)
- tempIncr: 0.1 (cuanto es capaz de cambiar la temperatura en 1 unidad de tiempo)

Users Parameters:

	InitialDistance ToAC	finalDistance ToAC	Pace	Movement	minComfort Temp	maxComfort Temp	Reliable Connection
User1	13.0	2.0	0.5	oneway	22.0	24.0	0.99
User2	15.0	3.0	0.5	oneway	18.0	23.0	0.99
User3	17.0	4.0	0.5	oneway	23.0	25.0	0.99

Results for scenario 1.3.3, CRISP VERSION

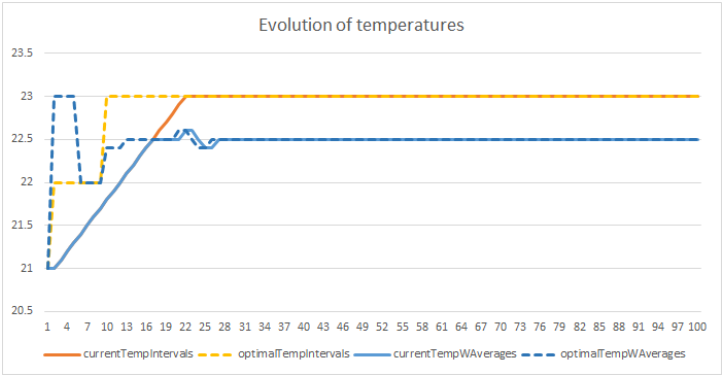


Fig. 16 Scenario 3. Evolution Temperatures

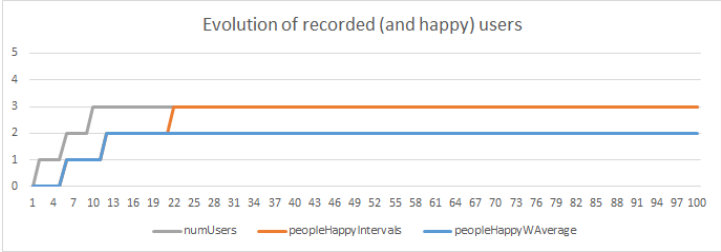


Fig. 17 Scenario 3. Evolution Recorded Users

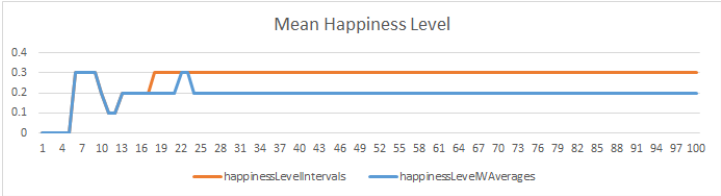


Fig. 18 Scenario 3. Mean Happiness Level

1.3.4 Scenario 4 ("MFB-Simple-noOverlap-ANTIGUO")

Igual que el anterior pero ponemos rangos de temperaturas que nunca se solapan de tal forma que vemos cómo se comportan los algoritmos.

AC controller Parameters:

- StandByTemp: 21.0 (temperatura en reposo)
- MaxDistance: 12.0 (podemos suponer que esto también representa el tamaño de la sala)
- tempIncr: 0.1 (cuanto es capaz de cambiar la temperatura en 1 unidad de tiempo)

Users Parameters:

Table 3 Scenario 4.

	InitialDistance ToAC	finalDistance ToAC	Pace	Movement	minComfort Temp	maxComfort Temp	Reliable Connection
User1	13.0	2.0	0.5	oneway	23.0	24.0	0.99
User2	15.0	2.0	0.5	oneway	25.0	26.0	0.99
User3	17.0	2.0	0.5	oneway	27.0	28.0	0.99

Results for scenario 1.3.4, CRISP VERSION

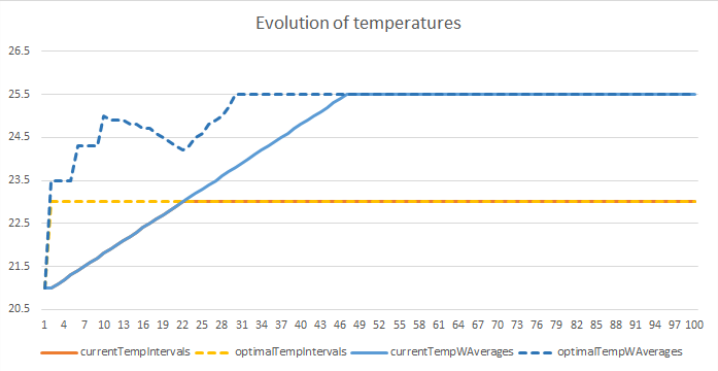


Fig. 19 Scenario 4. Evolution of Temperatures

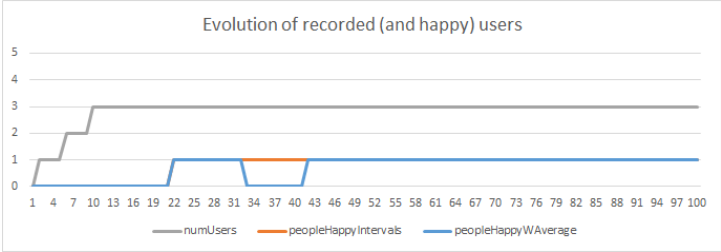


Fig. 20 Scenario 4. Evolution Recorded Users

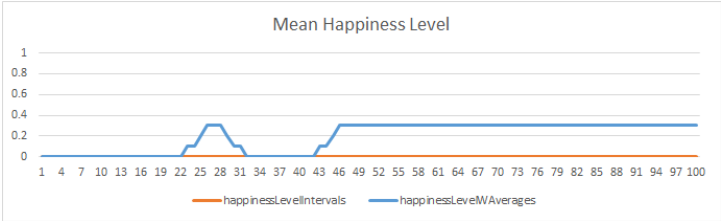


Fig. 21 Scenario 4. Mean Happiness Level

1.3.5 Scenario 4bis ("MFB-Simple-noOverlap-NUEVO")

Igual que el anterior pero ponemos rangos de temperaturas que nunca se solapan de tal forma que vemos cómo se comportan los algoritmos.

AC controller Parameters:

- StandByTemp: 21.0 (temperatura en reposo)
- MaxDistance: 12.0 (podemos suponer que esto también representa el tamaño de la sala)
- tempIncr: 0.1 (cuanto es capaz de cambiar la temperatura en 1 unidad de tiempo)

Users Parameters:

Table 4 Scenario 4.

	InitialDistance ToAC	finalDistance ToAC	Pace	Movement	minComfort Temp	maxComfort Temp	Reliable Connection
User1	12.0	2.0	0.5	oneway	21.0	22.0	0.99
User2	22.0	2.0	0.5	oneway	23.0	24.0	0.99
User3	32.0	2.0	0.5	oneway	25.0	26.0	0.99

Results for scenario 1.3.4bis, CRISP VERSION

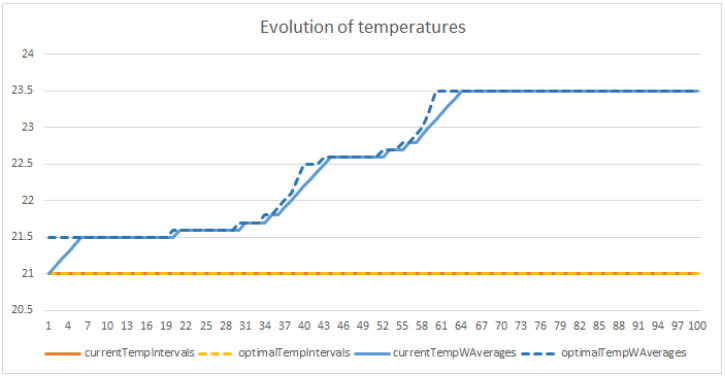


Fig. 22 Scenario 4bis. Evolution of Temperatures

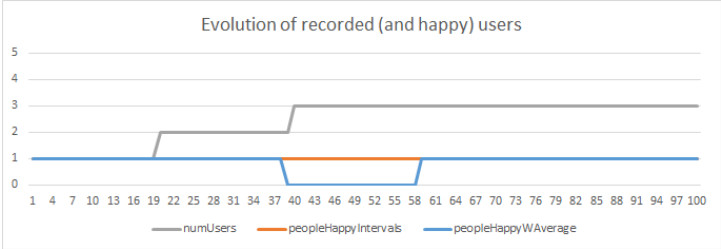


Fig. 23 Scenario 4bis. Evolution Recorded Users

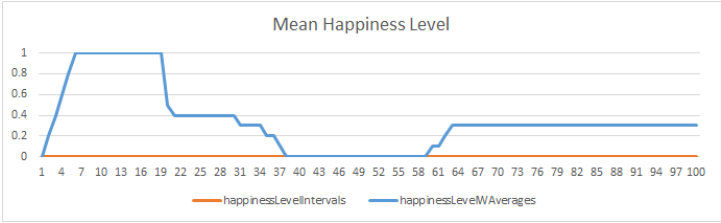


Fig. 24 Scenario 4bis. Mean Happiness Level

1.3.6 Scenario 5 ("MFB-Entran y salen")

Escenario donde hay 4 usuarios, 2 permanecen, uno está dentro y va saliendo y otro está fuera y entra.

AC controller Parameters:

- StandByTemp: 21.0 (temperatura en reposo)
- MaxDistance: 12.0 (podemos suponer que esto también representa el tamaño de la sala)
- tempIncr: 0.1 (cuanto es capaz de cambiar la temperatura en 1 unidad de tiempo)

Users Parameters:

Table 5 Scenario 4.

	InitialDistance ToAC	finalDistance ToAC	Pace	Movement	minComfort Temp	maxComfort Temp	Reliable Connection
User1	2.0	2.0	0.0	still	22.0	24.0	0.99
User2	4.0	4.0	0.0	still	18.0	23.0	0.99
User3	3.0	24.0	0.5	oneway	23.0	25.0	0.99
User4	20.0	3.0	0.5	nonstop	24.0	30.0	0.99

Results for scenario 1.3.5, CRISP VERSION

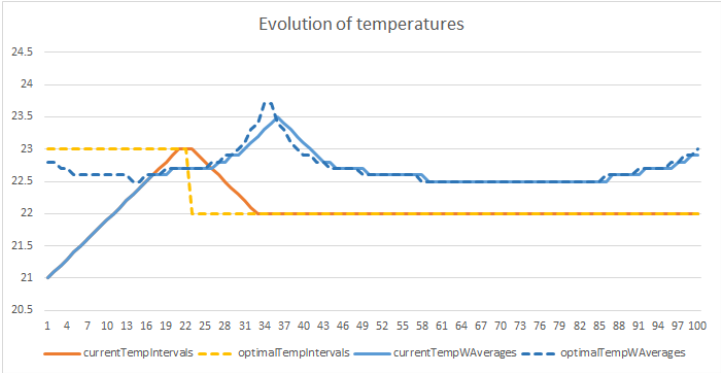


Fig. 25 Scenario 5. Evolution of Temperatures

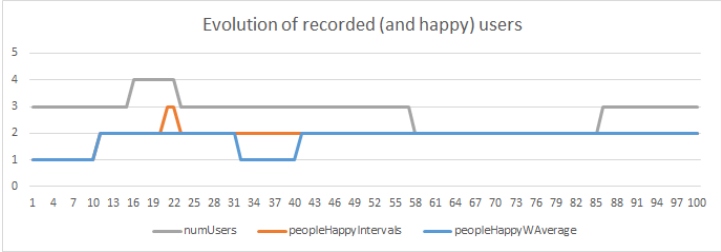


Fig. 26 Scenario 5. Evolution Recorded Users

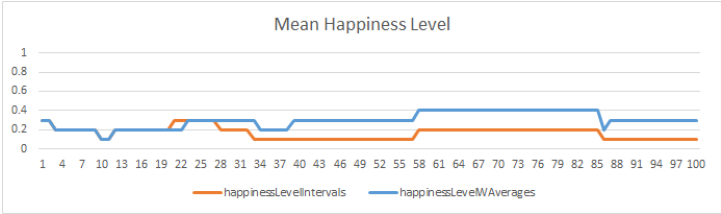


Fig. 27 Scenario 5. Mean Happiness Level

1.3.7 Scenario 6 ("MFB-Entran y salen")

Escenario donde hay 4 usuarios, 2 permanecen con valores razonables, dos non-stop uno con valores muy altos y otro muy bajos.

AC controller Parameters:

- StandByTemp: 21.0 (temperatura en reposo)
- MaxDistance: 12.0 (podemos suponer que esto también representa el tamaño de la sala)
- tempIncr: 0.1 (cuanto es capaz de cambiar la temperatura en 1 unidad de tiempo)

Users Parameters:

Table 6 Scenario 6.

	InitialDistance ToAC	finalDistance ToAC	Pace	Movement	minComfort Temp	maxComfort Temp	Reliable Connection
User1	2.0	2.0	0.0	still	22.0	24.0	0.99
User2	4.0	4.0	0.0	still	18.0	28.0	0.99
User3	3.0	20.0	0.5	nonstop	12.0	19.0	0.99
User4	20.0	3.0	0.5	nonstop	27.0	34.0	0.99

Results for scenario 1.3.6, CRISP VERSION

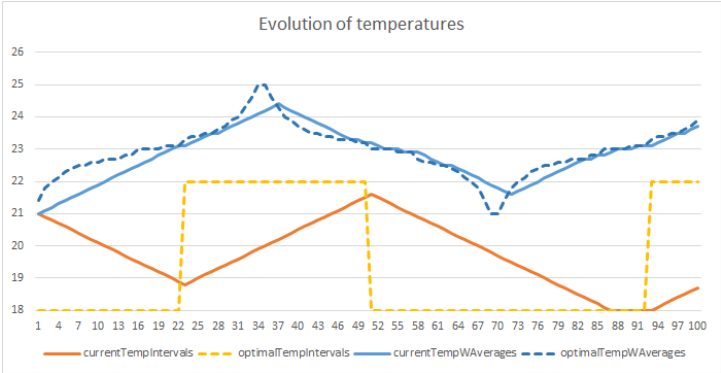


Fig. 28 Scenario 6. Evolution of Temperatures

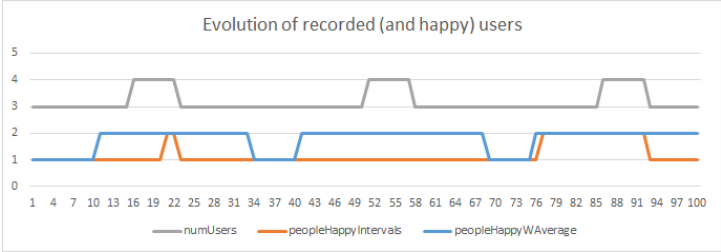


Fig. 29 Scenario 6. Evolution Recorded Users

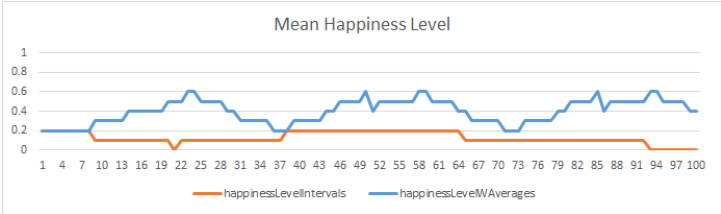


Fig. 30 Scenario 6. Mean Happiness Level

1.3.8 Scenario 7 ("MFB-estable")

Escenario donde hay 4 usuarios quietos y de acuerdo, lo unico es que tienen conexiones poco fiables.

AC controller Parameters:

- StandByTemp: 21.0 (temperatura en reposo)
- MaxDistance: 12.0 (podemos suponer que esto también representa el tamaño de la sala)
- tempIncr: 0.1 (cuanto es capaz de cambiar la temperatura en 1 unidad de tiempo)

Users Parameters:

Table 7 Scenario 7.

	InitialDistance ToAC	finalDistance ToAC	Pace	Movement	minComfort Temp	maxComfort Temp	Reliable Connection
User1	2.0	2.0	0.0	still	22.0	26.0	0.5
User2	2.0	2.0	0.0	still	23.0	27.0	0.5
User3	2.0	2.0	0.0	still	24.0	28.0	0.5
User4	2.0	2.0	0.0	still	25.0	29.0	0.5

Results for scenario 1.3.7, CRISP VERSION

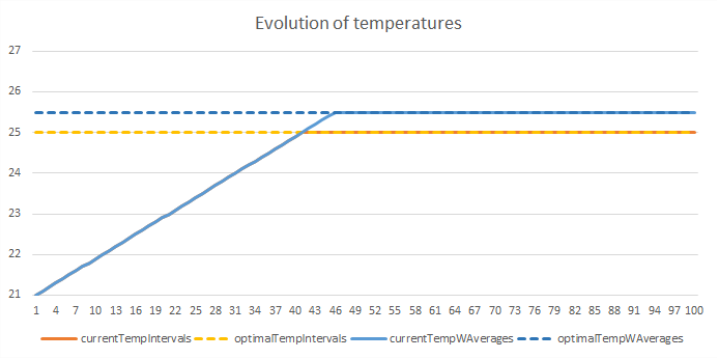


Fig. 31 Scenario 7. Evolution of Temperatures

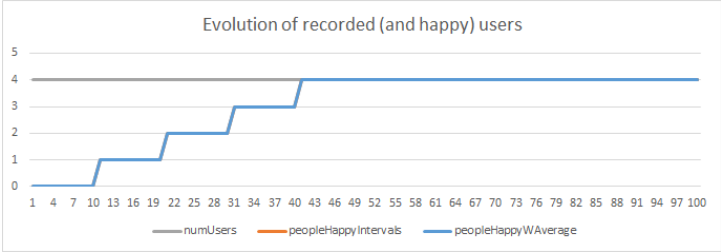


Fig. 32 Scenario 7. Evolution Recorded Users

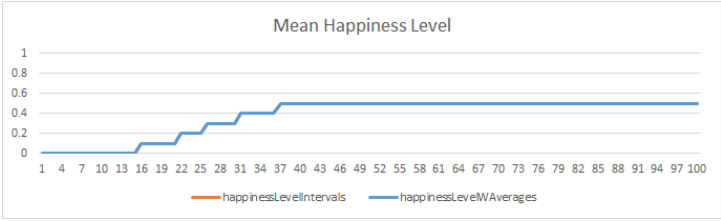


Fig. 33 Scenario 7. Mean Happiness Level

1.3.9 Scenario 8 ("MFB-Mess")

Escenario donde hay varios usuarios que pasan o salen y entran.

AC controller Parameters:

- StandByTemp: 21.0 (temperatura en reposo)
- MaxDistance: 12.0 (podemos suponer que esto también representa el tamaño de la sala)
- tempIncr: 0.1 (cuanto es capaz de cambiar la temperatura en 1 unidad de tiempo)

Users Parameters:

Table 8 Scenario 8.

	InitialDistance ToAC	finalDistance ToAC	Pace	Movement	minComfort Temp	maxComfort Temp	Reliable Connection
User1	1.0	15.0	0.2	nonstop	32.0	36.0	0.5
User2	20.0	10.0	0.5	nonstop	18.0	22.0	0.5
User3	15.0	9.0	0.2	nonstop	24.0	28.0	0.5
User4	4.0	10.0	0.5	nonstop	16.0	20.0	0.5
User4	1.0	12.0	0.5	nonstop	23.0	25.0	0.7
User6	14.0	10.0	0.1	nonstop	17.0	27.0	0.8
User7	15.0	9.0	0.5	nonstop	12.0	14.0	0.9
User8	4.0	10.0	0.5	nonstop	20.0	22.0	0.9

Results for scenario 1.3.8, CRISP VERSION

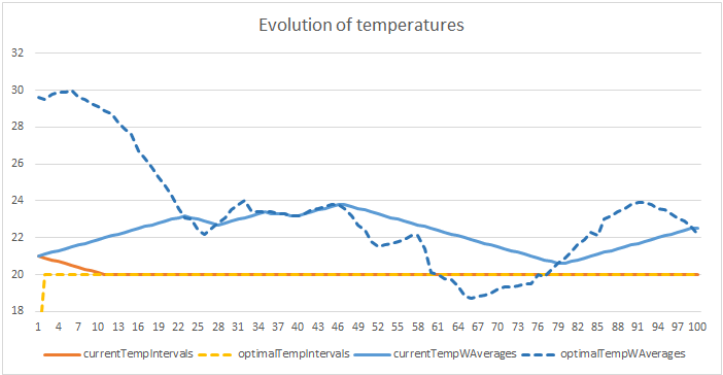


Fig. 34 Scenario 8. Evolution of Temperatures

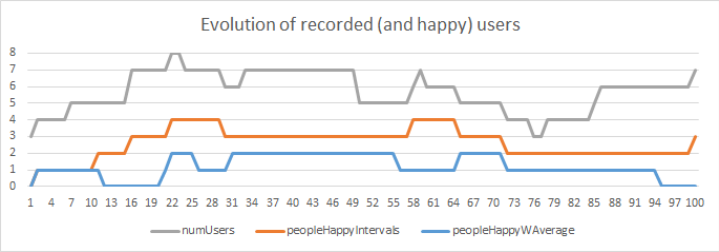


Fig. 35 Scenario 8. Evolution Recorded Users

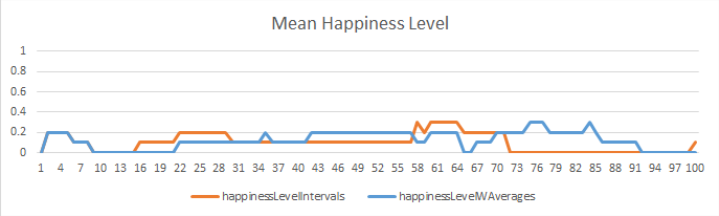


Fig. 36 Scenario 8. Mean Happiness Level

2 Adding Measurement Uncertainty

2.1 Structure of the system

The next step is to add *measurement uncertainty* to the system. This type of aleatory uncertainty represents the (lack of) precision in the measurement devices, as well as the level of confidence in the OCL expressions, which no longer correspond to the strict Boolean dichotomy true-false, but allow them to be "partially" true or false (by means of real values in the range [0,1] that represent the "confidence" that we assign to the truthfulness of an OCL value or expression. Type `UBoolean` extends type Boolean with such confidence, and therefore `UBoolean(true,0.9)` means that we are only 90% sure of a given proposition. Precision in attributes representing Real numbers is expressed by extending them with the standard deviation of their possible values, as customary in other engineering disciplines dealing with physical quantities. Thus, values of type UReal are pairs (x,u) where x is the expected value, and u its standard uncertainty – e.g., `UReal(3.0,0.1)` represents the number 3.0 ± 0.1.

The AC system with this new information can be modeled (including the Test class) as follows:

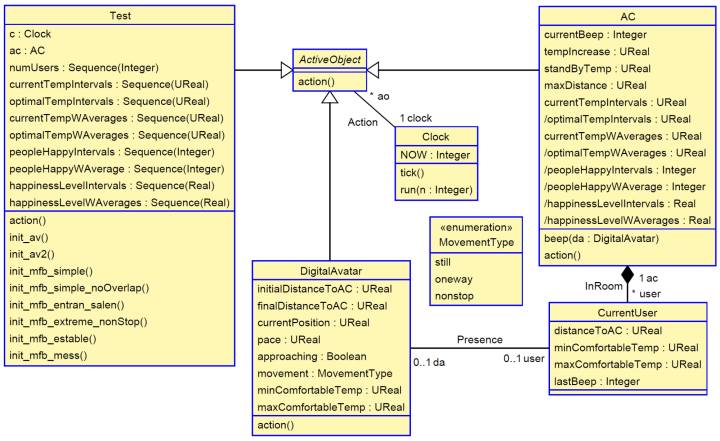


Fig. 37 AC system including Test class.

2.2 Scenario results for the measurement uncertainty (MU) version

2.2.1 Results for scenario MU.1, "MU-AV"

Scenario with five users, all inside the room at all times. 2 still users, 1 approaching the AC, 1 leaving it, one moving up and down.

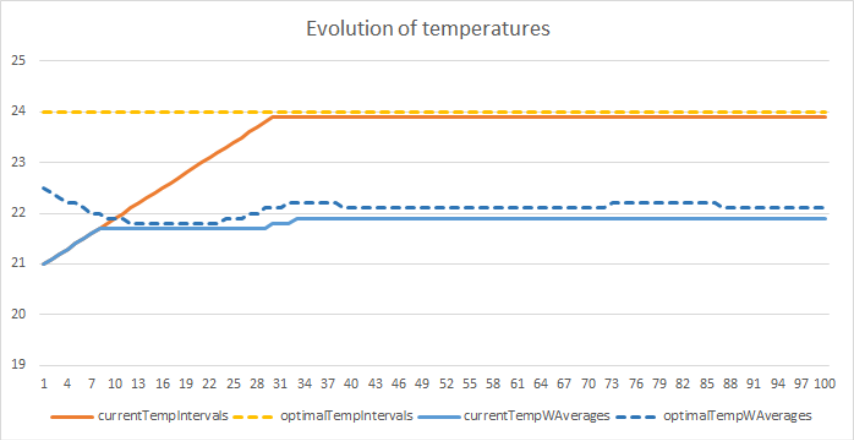


Fig. 38 MU-AV Evolution of temperature

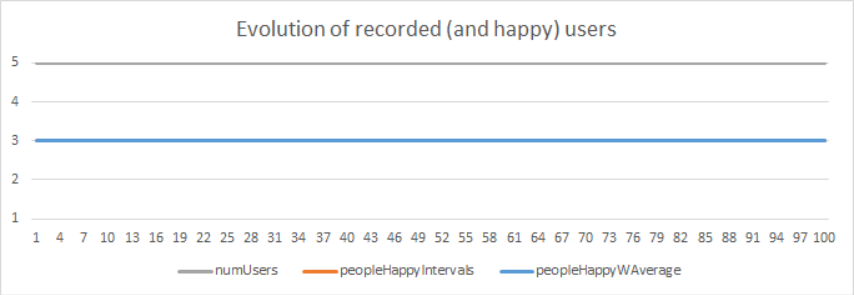


Fig. 39 MU-AV Recorded and Happy Users

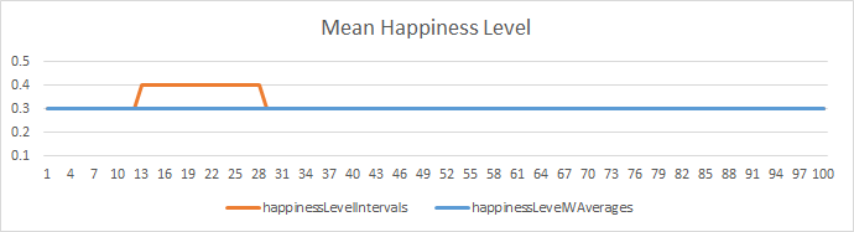


Fig. 40 MU-AV Happiness Level

It is interesting to compare them with the ones obtained for the "crisp" system. For example, the new ones are smoother and represent temperature evolution in a more natural manner. Moreover, the users" happiness is slightly lower because... CHECK THIS OUT... some of the users' temperature comfort intervals that did not intersect before but that were very close (≈ 0.1 °C) now are considered to have some likelihood to overlap. Note that this is what happens in reality, where a typical human being cannot easily differentiate between 21.9 and 22.0.

2.2.2 Results for scenario MU.2, "MU-AV2"

Five users, 3 of them inside the room at all times. 2 still users, 1 approaching the AC starting from outside the room, 1 leaving the room, one moving up and down, continuously entering and leaving the room.

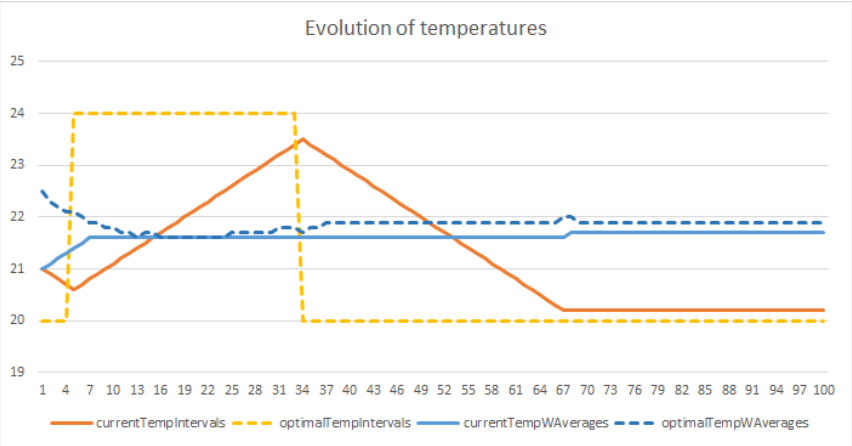


Fig. 41 MU-AV2 Evolution of temperature

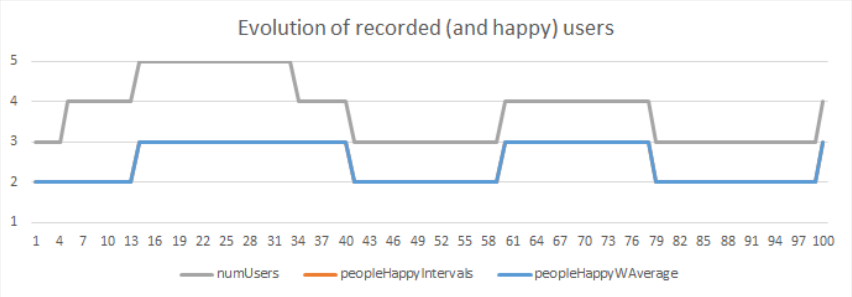


Fig. 42 MU-AV2 Recorded and Happy Users

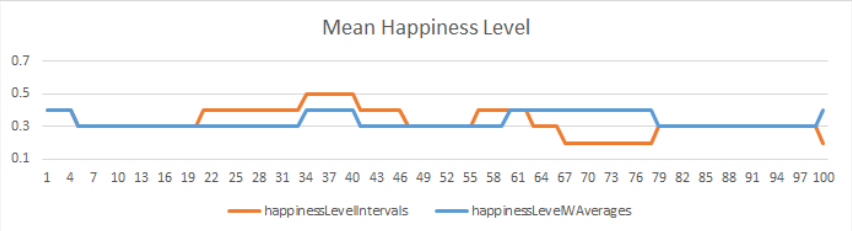


Fig. 43 MU-A2V Happiness Level

2.2.3 Results for scenario MU.3, "MU-MFB-Simple"

Scenario where 3 users arrive gradually with moderate values that matching at 23.0 ºC. They all stay inside the room.

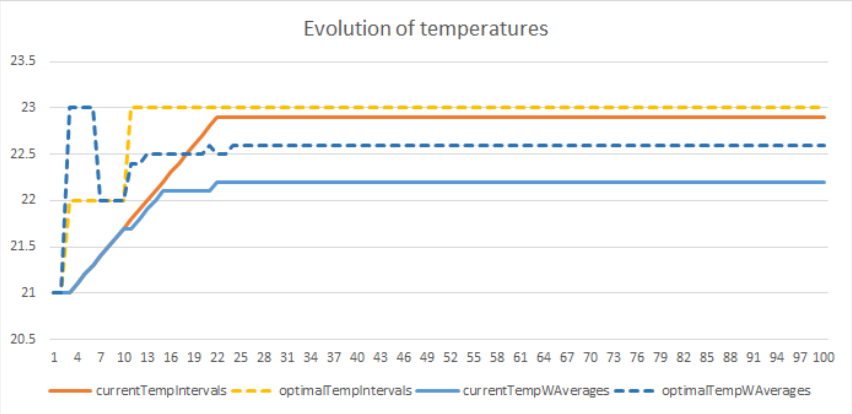


Fig. 44 MU-MFB-Simple Evolution of temperature

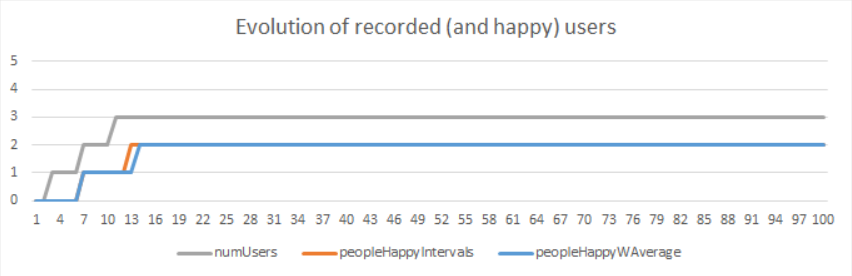


Fig. 45 MU-MFB-Simple Recorded and Happy Users

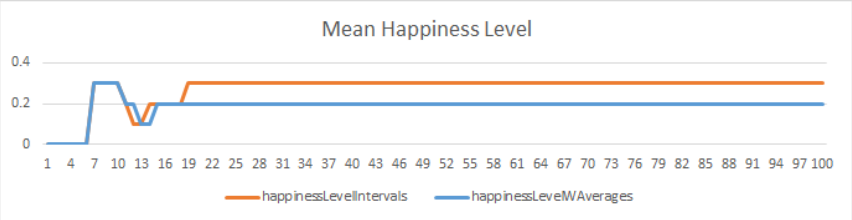


Fig. 46 MU-MFB-Simple Happiness Level

2.2.4 Results for scenario MU.4, "MU-No-Overlap-V0"

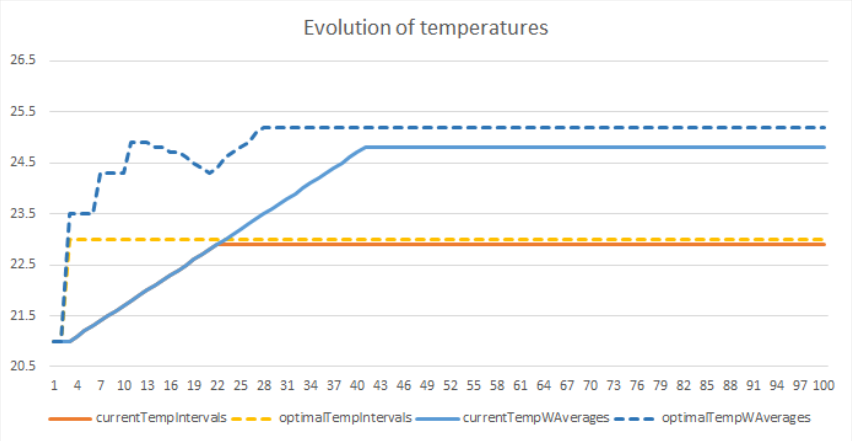


Fig. 47 MU-No-Overlap-V0 Evolution of temperature

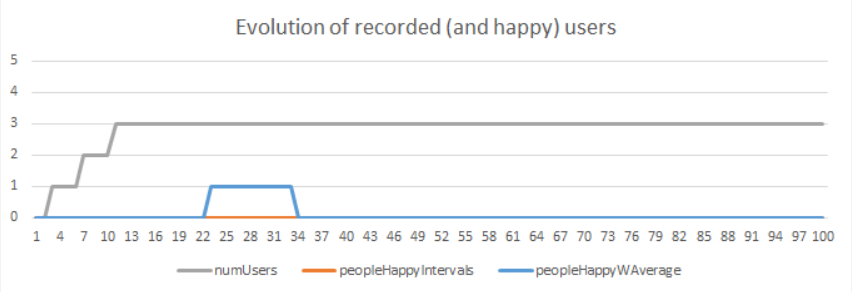


Fig. 48 MU-No-Overlap-V0 Recorded and Happy Users

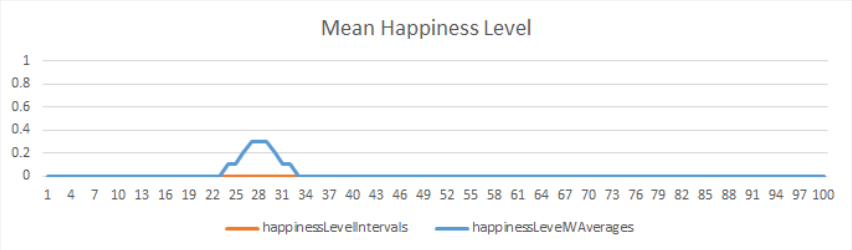


Fig. 49 MU-No-Overlap-V0 Happiness Level

2.2.5 Results for scenario MU.4B, "MU-No-Overlap-V1"

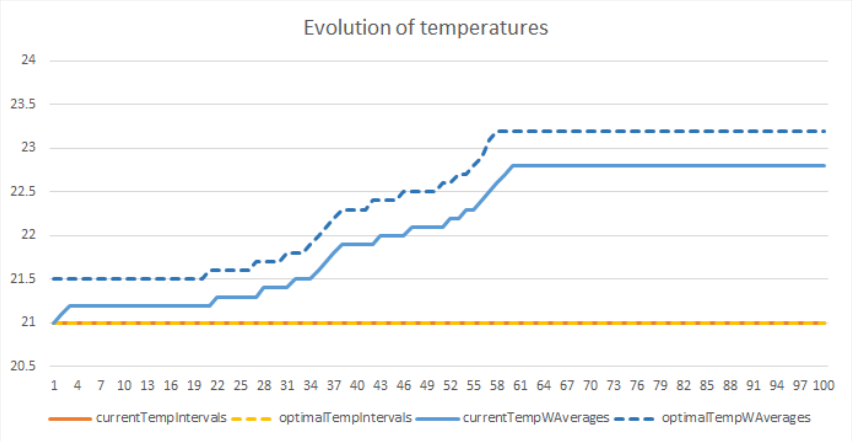


Fig. 50 MU-No-Overlap-V1 Evolution of temperature

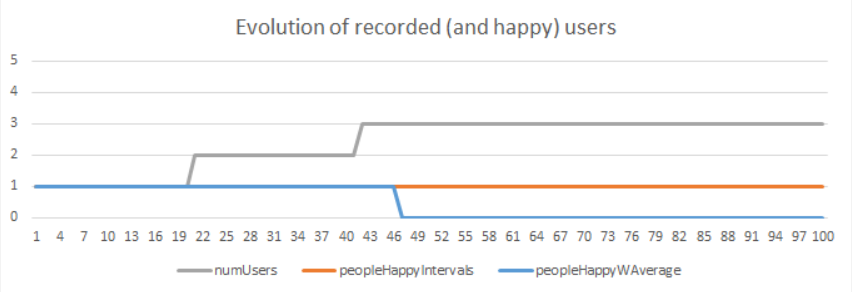


Fig. 51 MU-No-Overlap-V1 Recorded and Happy Users

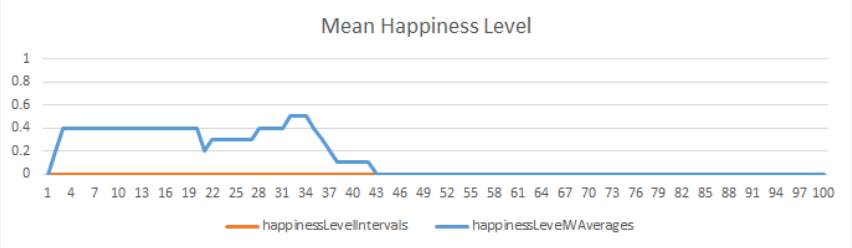


Fig. 52 MU-No-Overlap-V1 Happiness Level

2.2.6 Results for scenario MU.5, "MU-In-and-Out"

Scenario where there are 4 users, 2 stay, one is inside and goes out and another is outside and comes in.

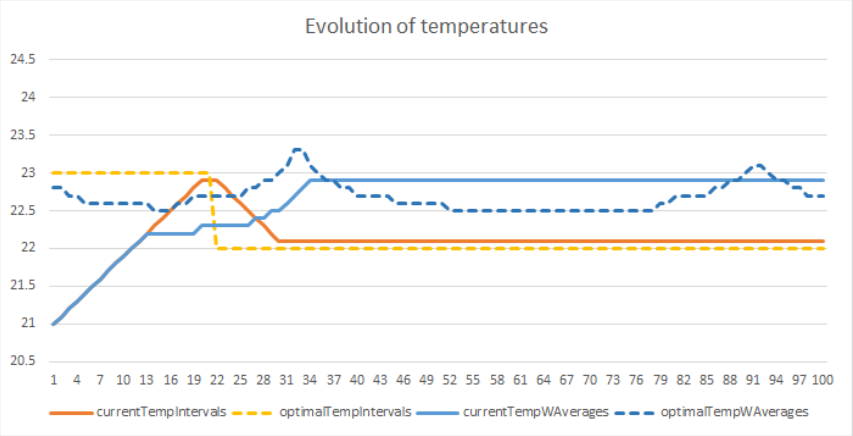


Fig. 53 MU-In-and-Out Evolution of temperature

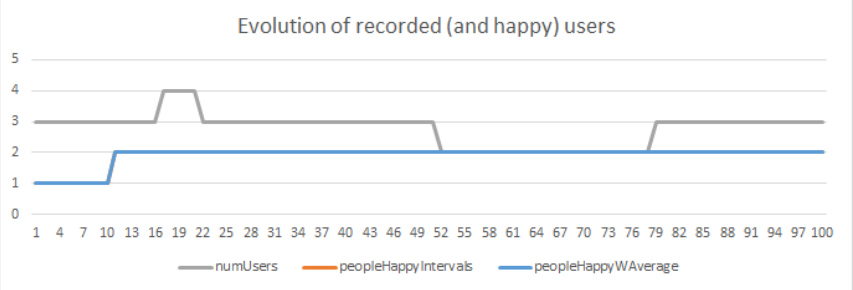


Fig. 54 MU-In-and-Out Recorded and Happy Users

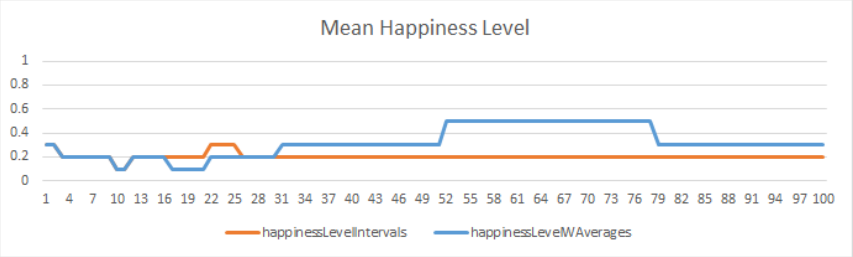


Fig. 55 MU-In-and-Out Happiness Level

2.2.7 Results for scenario MU.6, "MU-Extreme-Non-Stop"

Scenario where there are 4 users: 2 stay inside with moderate values and two non-stop, one of them with very high values and the other with very low values.

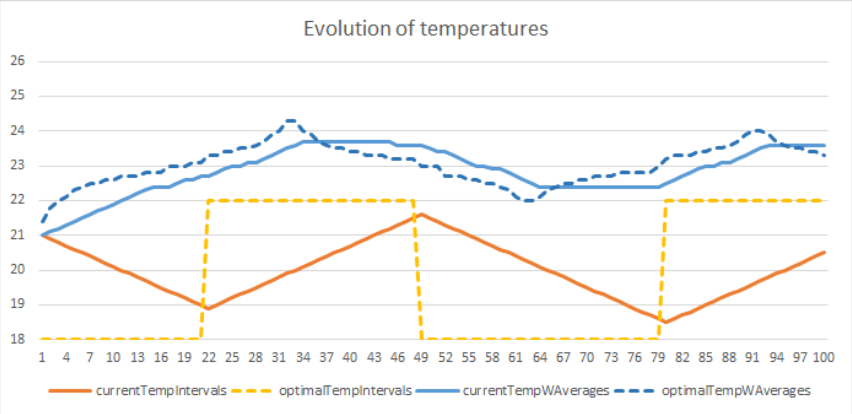


Fig. 56 MU-Extreme-Non-Stop Evolution of temperature

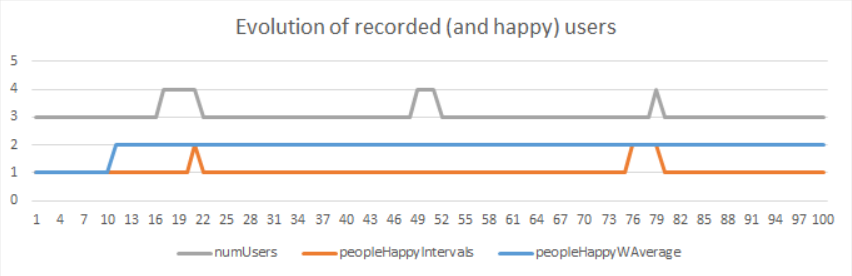


Fig. 57 MU-Extreme-Non-Stop Recorded and Happy Users

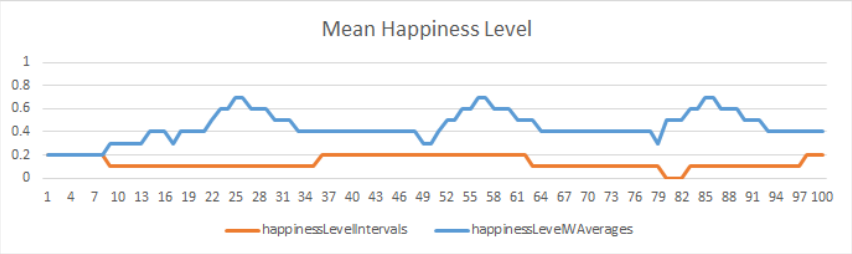


Fig. 58 MU-Extreme-Non-Stop Happiness Level

2.2.8 Results for scenario MU.7, "MU-Unreliable-Conex"

Scenario where there are 4 users inside and with similar and moderate values but their connections are unreliable.

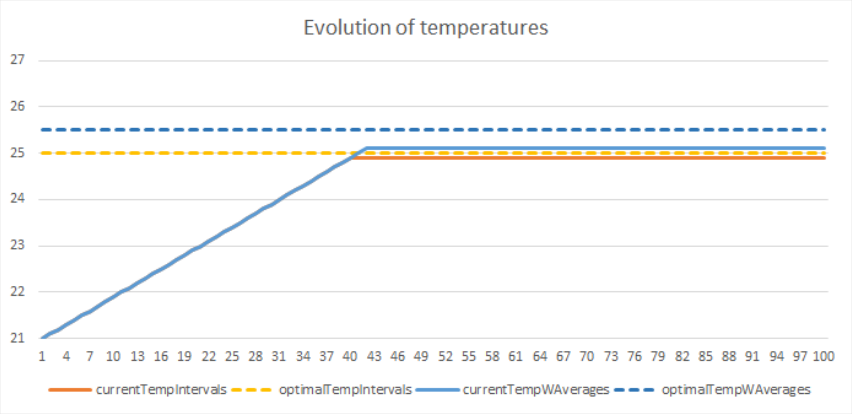


Fig. 59 MU-Unreliable-Conex Evolution of temperature

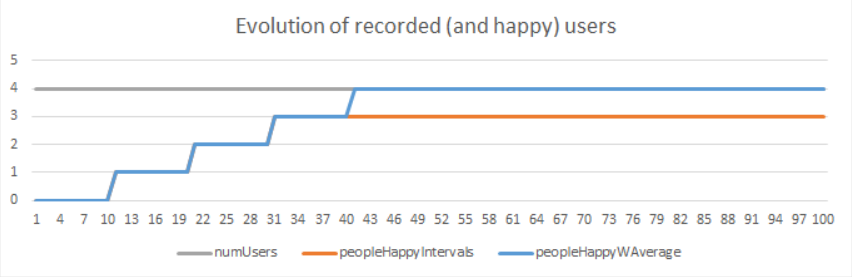


Fig. 60 MU-Unreliable-Conex Recorded and Happy Users

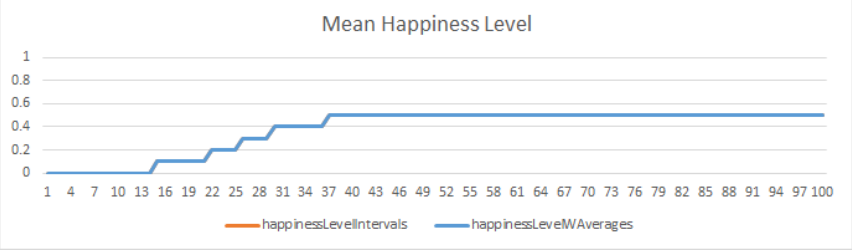


Fig. 61 MU-Unreliable-Conex Happiness Level

2.2.9 Results for scenario MU.8, "MU-Mess"

Scenario with 8 users entering, leaving or passing by.

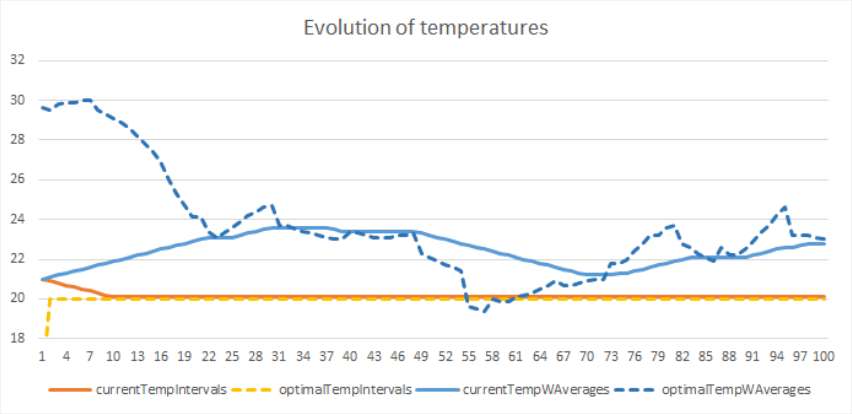


Fig. 62 MU-Mess Evolution of temperature

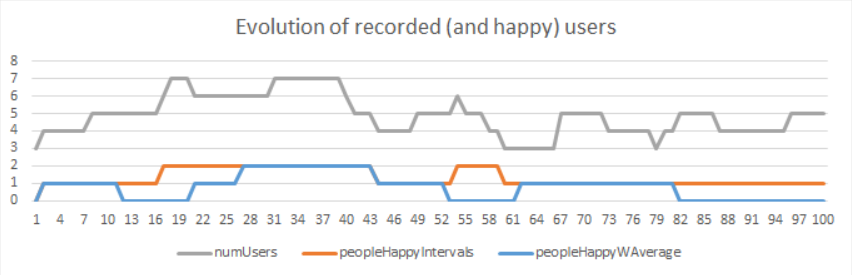


Fig. 63 MU-Mess Recorded and Happy Users

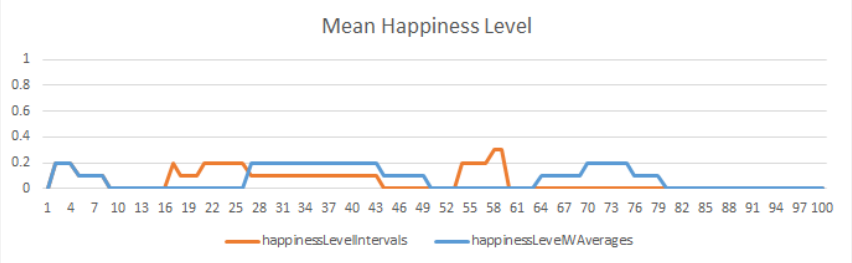


Fig. 64 MU-Mess Happiness Level

3 Adding Occurrence Uncertainty

3.1 Structure of the system

Occurrence uncertainty refers to the likelihood of existence of the physical elements captured by the model. It is normally represented by a Real number in the range [0..1] that expresses the confidence that we have that the element represented by a given instance or a given link in an object diagram actually exists in reality.

In this case, we have modeled this type of uncertainty using two attributes, called *confidence* and *reliableConnection*, added to the `CurrentUser` and `DigitalAvatar` classes, respectively.

The first one refers to the probability that the user of the digital avatar is indeed in the room, and the second one refers to the probability that the Bluetooth connection is working. The value of this latter attribute normally depends on the quality of the device, and it is individually defined when creating the instances of the digital avatars. Attribute confidence, however, is computed for the current users depending on the last time that the corresponding digital avatar beeped, and on how trustful the comfort range of the user looks like. More precisely, the following OCL expression specifies the value of attribute confidence:

```
confidence : Real derive:
-- two factors influence the confidence: the last time it beeped,
-- and how sensible its comfort zone is
 ( if self.ac.currentBeep - self.lastBeep <= 2
   then 1.0
   else if self.ac.currentBeep - self.lastBeep = 3
        then 0.7 else 0.2 endif
   endif) *
 ( if ((self.minComfortableTemp>=UReal(27.0,0.1)).confidence>=0.95
       or
       (self.maxComfortableTemp<=UReal(19.0,0.1)).confidence>=0.95
           ↪ )
   then 0.0
   else
       let below:UReal = (UReal(19.0,0.1)-self.minComfortableTemp)
           ↪.max(UReal(0.0,0.1)) in
       let above:UReal = (self.minComfortableTemp-UReal(27.0,0.1))
           ↪.max(UReal(0.0,0.1)) in
       1.0 - ((below+above)/(self.maxComfortableTemp-self.
           ↪minComfortableTemp)).value()
  endif
 )
```

With this, the model (see Fig. 65) with occurrence uncertainty is the same as the previous one, but with the two new attributes (*confidence* and *reliableConnection*) added.

The behavior of the system does not change, apart from the following changes:

- Only users with a confidence >=0.05 are considered

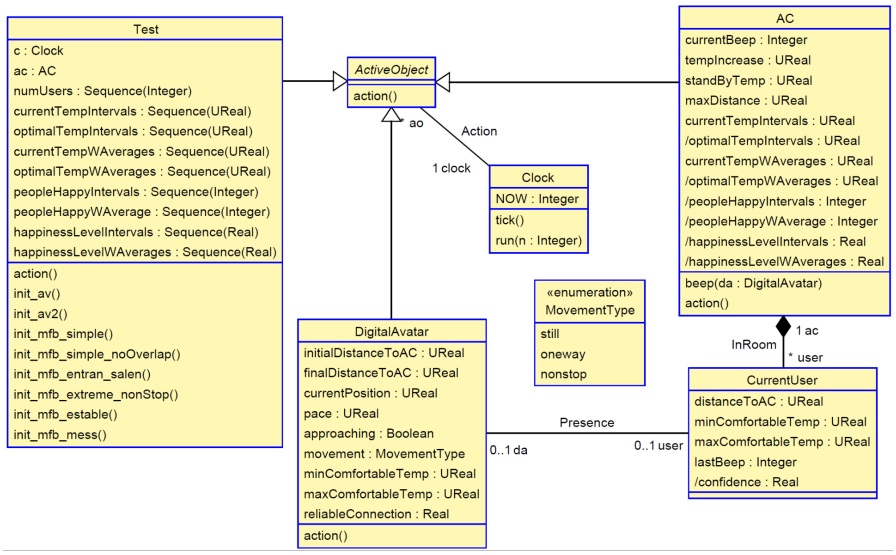


Fig. 65 Air Conditioning Model adding occurrence uncertainty

- When calculating the optimal temperatures, users are weighed according to their respective confidences.

This way, when the reliability of the connection is 1.00, the behavior of the system coincides with that of the previous MU-Occ.version (i.e., with measurement uncertainty).

3.2 Scenario Results for Measurement and Occurrence Uncertainties (MU-Occ.Occ) Version

3.2.1 Results for scenario MU-Occ.1, "MU-Occ.AV"

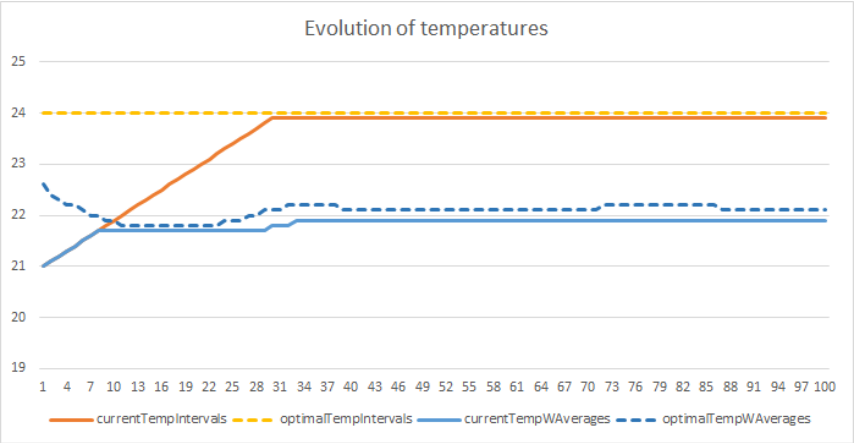


Fig. 66 MU-Occ.AV Evolution of temperature

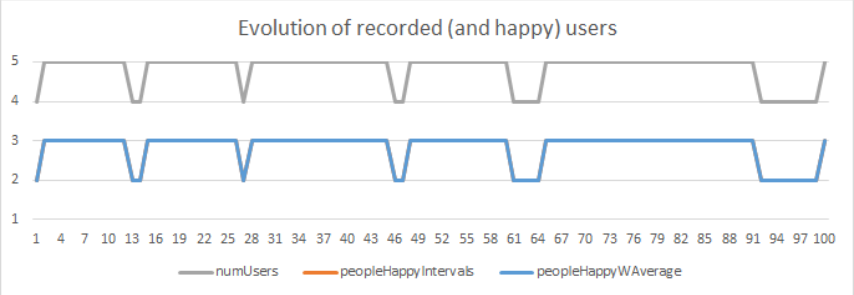


Fig. 67 MU-Occ.AV Recorded and Happy Users

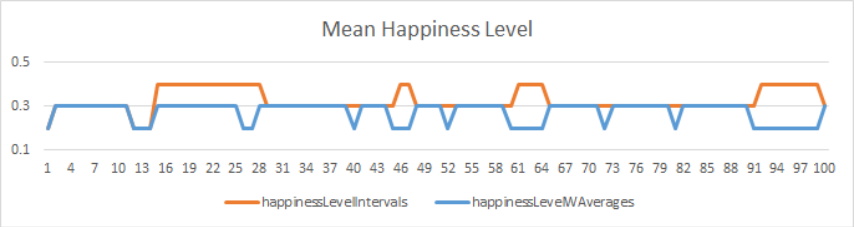


Fig. 68 MU-Occ.AV Happiness Level

3.2.2 Results for scenario MU-Occ.2, "MU-Occ.AV2"

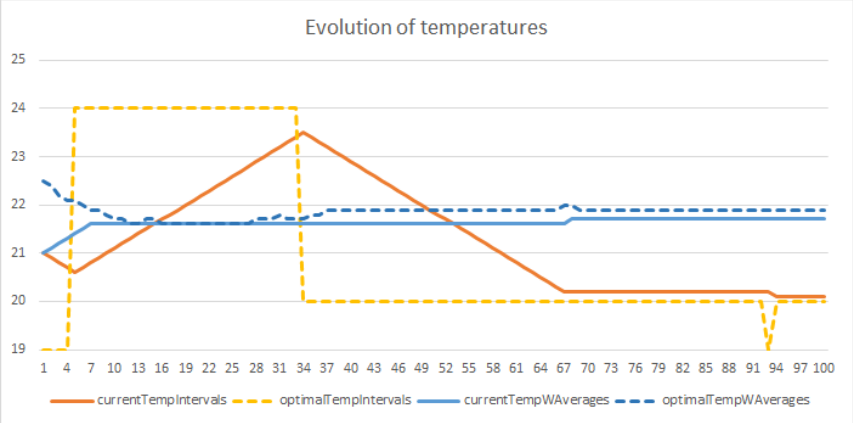


Fig. 69 MU-Occ.AV2 Evolution of temperature

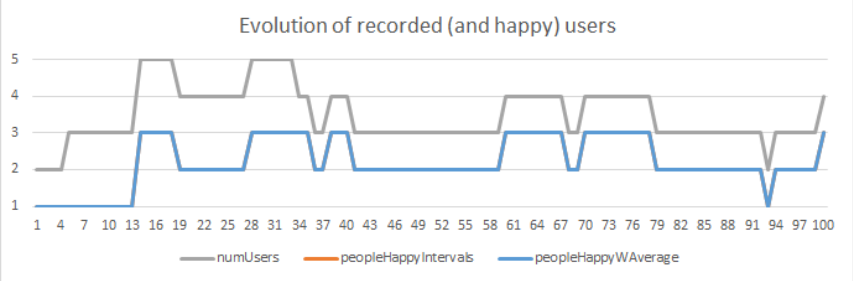


Fig. 70 MU-Occ.AV2 Recorded and Happy Users

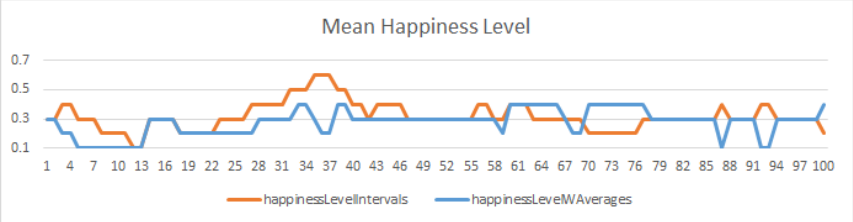


Fig. 71 MU-Occ.A2V Happiness Level

3.2.3 Results for scenario MU-Occ.3, "MU-Occ.MFB-Simple"

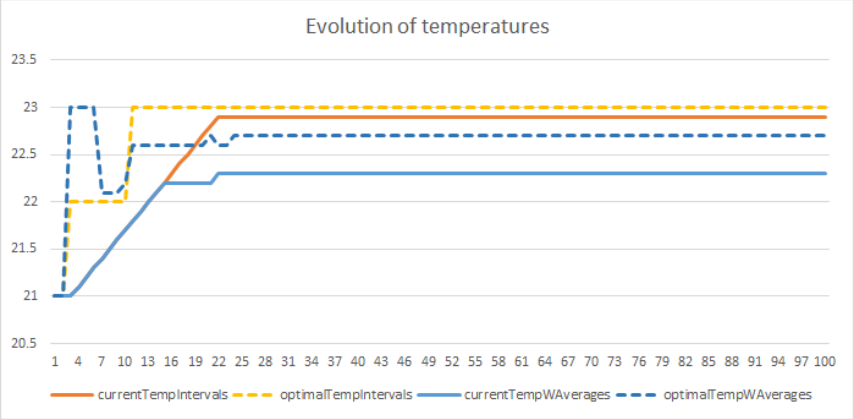


Fig. 72 MU-Occ.MFB-Simple Evolution of temperature

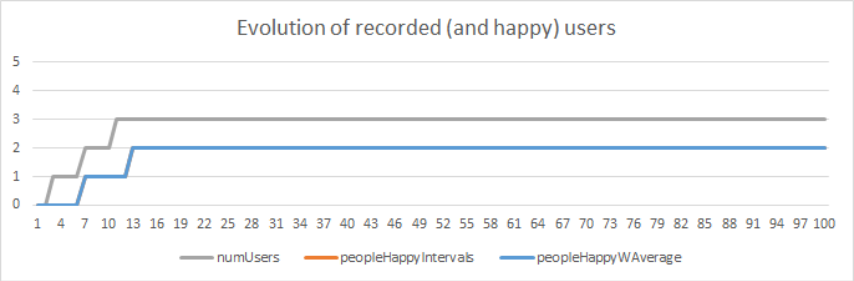


Fig. 73 MU-Occ.MFB-Simple Recorded and Happy Users

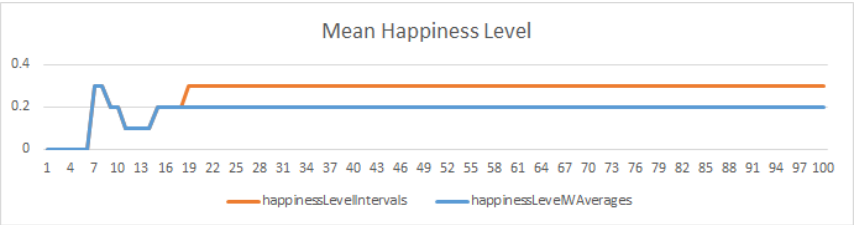


Fig. 74 MU-Occ.MFB-Simple Happiness Level

3.2.4 Results for scenario MU-Occ.4, "MU-Occ.No-Overlap-V0"

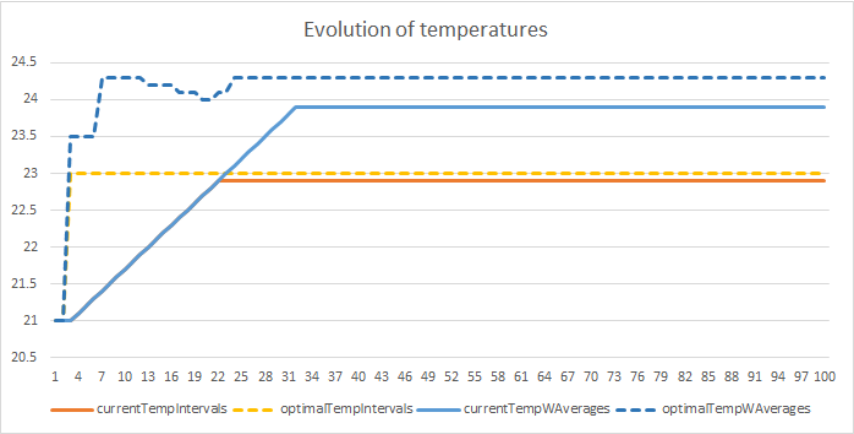


Fig. 75 MU-Occ.No-Overlap-V0 Evolution of temperature

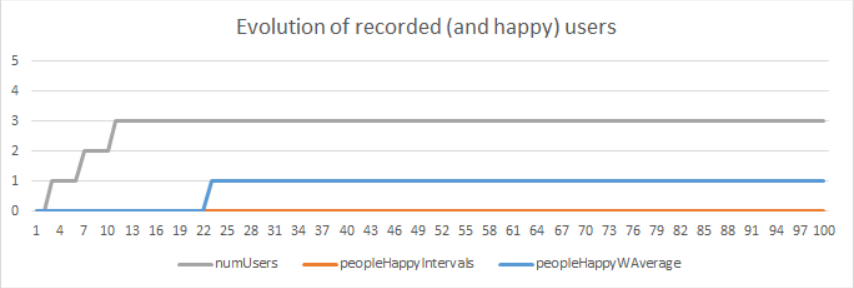


Fig. 76 MU-Occ.No-Overlap-V0 Recorded and Happy Users

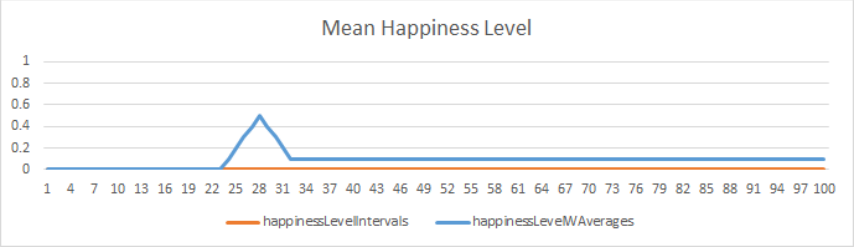


Fig. 77 MU-Occ.No-Overlap-V0 Happiness Level

3.2.5 Results for scenario MU-Occ.4B, "MU-Occ.No-Overlap-V1"

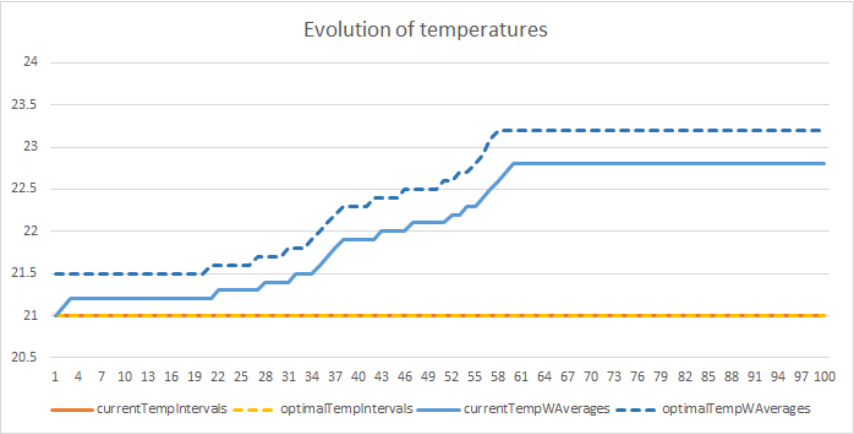


Fig. 78 MU-Occ.No-Overlap-V1 Evolution of temperature

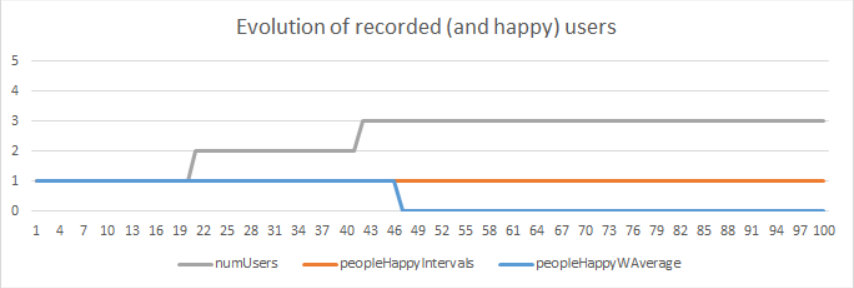


Fig. 79 MU-Occ.No-Overlap-V1 Recorded and Happy Users

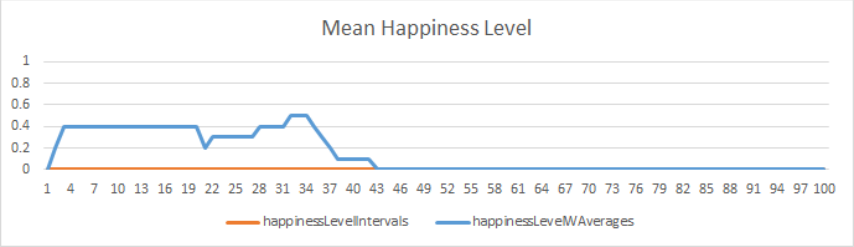


Fig. 80 MU-Occ.No-Overlap-V1 Happiness Level

3.2.6 Results for scenario MU-Occ.5, "MU-Occ.In-and-Out"

Scenario where there are 4 users, 2 stay, one is inside and goes out and another is outside and comes in.

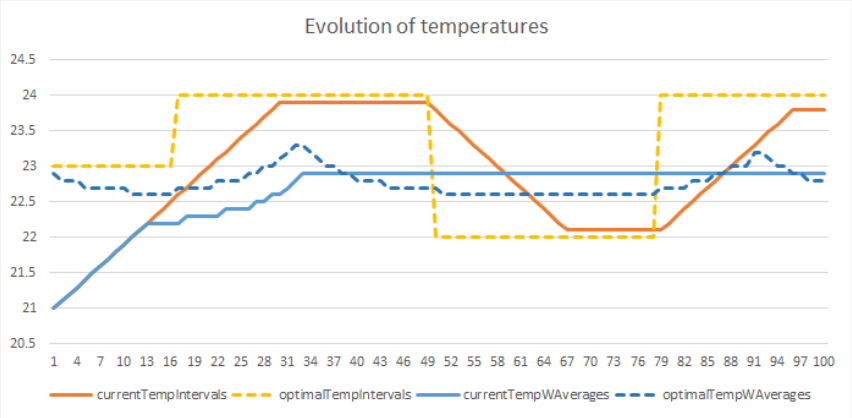


Fig. 81 MU-Occ.In-and-Out Evolution of temperature

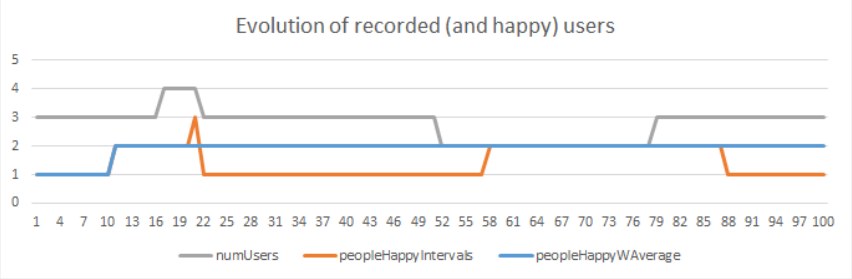


Fig. 82 MU-Occ.In-and-Out Recorded and Happy Users

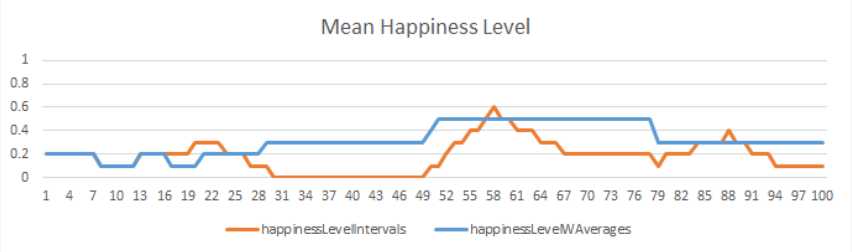


Fig. 83 MU-Occ.In-and-Out Happiness Level

3.2.7 Results for scenario MU-Occ.6, "MU-Occ.Extreme-Non-Stop"

Scenario where there are 4 users: 2 stay inside with moderate values and two non-stop, one of them with very high values and the other with very low values.

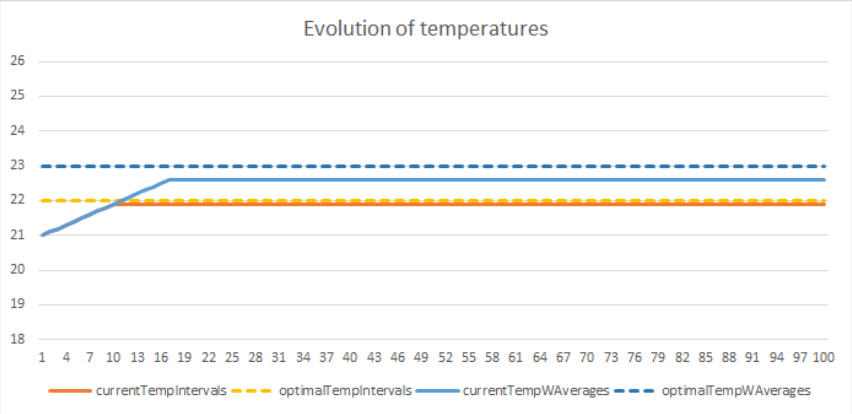


Fig. 84 MU-Occ.Extreme-Non-Stop Evolution of temperature

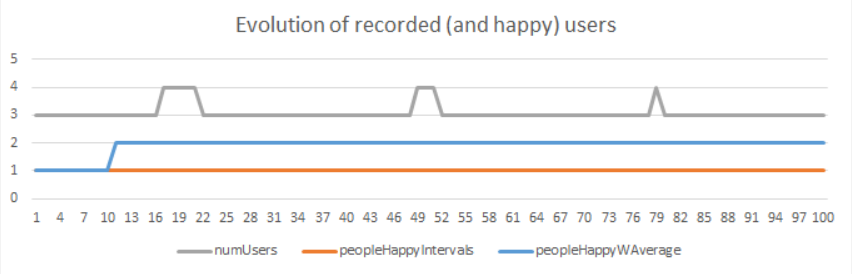


Fig. 85 MU-Occ.Extreme-Non-Stop Recorded and Happy Users

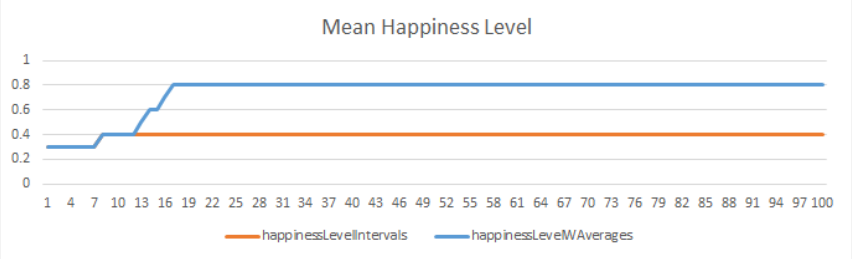


Fig. 86 MU-Occ.Extreme-Non-Stop Happiness Level

3.2.8 Results for scenario MU-Occ.7, "MU-Occ.Unreliable-Conex"

Scenario where there are 4 users inside and with similar and moderate values but their connections are unreliable.

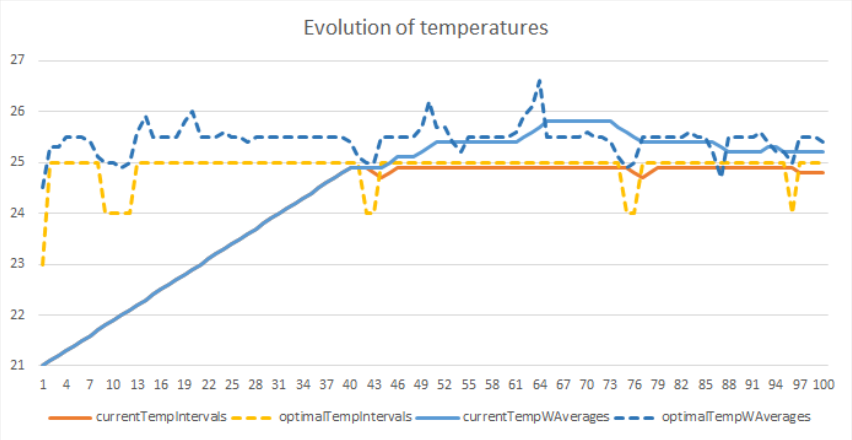


Fig. 87 MU-Occ.Unreliable-Conex Evolution of temperature

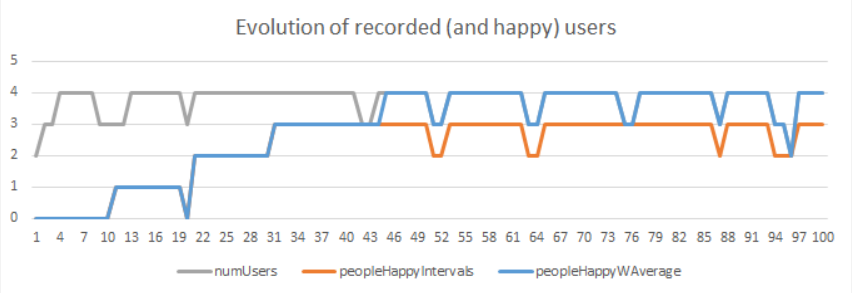


Fig. 88 MU-Occ.Unreliable-Conex Recorded and Happy Users

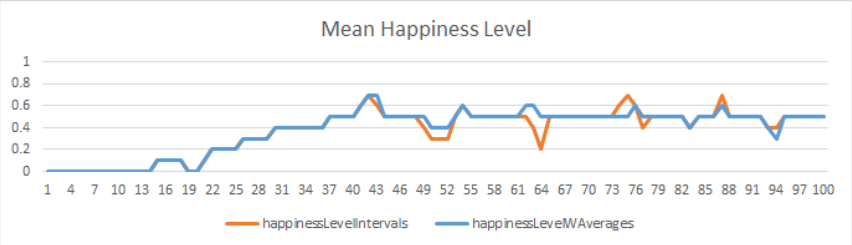


Fig. 89 MU-Occ.Unreliable-Conex Happiness Level

3.2.9 Results for scenario MU-Occ.8, "MU-Occ.Mess"

Scenario with 8 users entering, leaving or passing by.

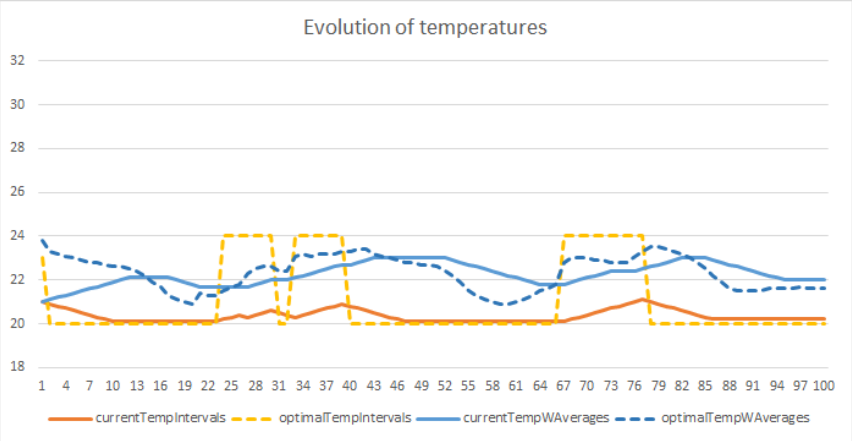


Fig. 90 MU-Occ.Mess Evolution of temperature

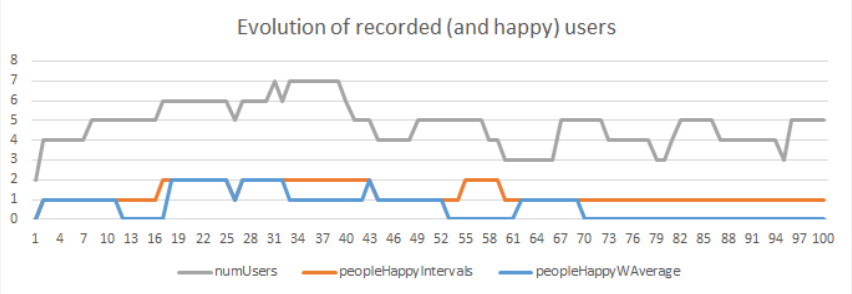


Fig. 91 MU-Occ.Mess Recorded and Happy Users

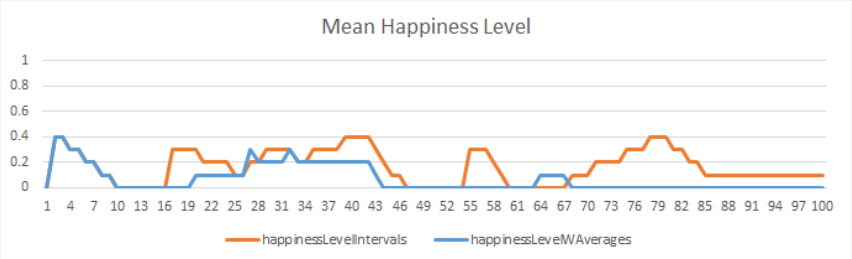


Fig. 92 MU-Occ.Mess Happiness Level

A Algorithms for deciding the temperature of the AC system

Antonio ▶ *Este anexo tengo que actualizarlo, ahora mismo no está actualizado.* ◀

A.1 Crisp and MU cases

The first algorithm (Intervals) computes the minimum value that makes more users happy (because it is inside in their comfort ranges). It can be specified is OCL as follows:

```
optimalTempIntervals : Real derive =
    if self.user->isEmpty then  -- no users in the room...
        self.standByTemp
    else
        let minmin : Integer = self.user->collect(
            ↪minComfortableTemp)->min().floor*10 in
        let maxmax : Integer = self.user->collect(
            ↪maxComfortableTemp)->max().ceil*10 in
        let maxUsers : Integer = Sequence{minmin..maxmax}->
            iterate(i; x:Sequence(Integer)=Sequence{}|x->append(
                ↪self.user->
            select(u|u.minComfortableTemp<=i/10 and
                    u.maxComfortableTemp>=i/10)->size()))->max()
                        ↪in
        Sequence{minmin..maxmax}->select(i|
                                    self.user->select(u|u.
                                        ↪minComfortableTemp<=i/10
                                        ↪ and
                                        u.maxComfortableTemp>=i
                                            ↪/10)->size()=
                                            ↪maxUsers)->min()/10
    endif
```

The second algorithm (WAverage) calculates a weighed average using as values the averages of the comfort ranges of each user, and the weights are calculated in inverse proportion to the square of the distance to the AC controller.

```
optimalTempWAverages : Real derive =
  if self.user->isEmpty then  --- no users in the room...
    self.standByTemp
  else
    let minDistance : Real = self.user->collect(distanceToAC*
        ↪distanceToAC)->min() in
    let weight : Real =
        self.user->collect(u| ((minDistance*minDistance) / (u.
            ↪distanceToAC*u.distanceToAC))) -> sum() in
    (self.user->collect(u|
            ((u.minComfortableTemp+u.maxComfortableTemp)/2) *
             (minDistance*minDistance/(u.distanceToAC*u.
                 ↪distanceToAC))/weight) -> sum())
  endif
```

The derived values of the rest of the attributes of class `AC` are specified in OCL as follows:

```
optimalTemp : Real derive = (optimalTempIntervals +
    ↪optimalTempWAverages)/2

standByTemp : Real init:20.0 -- when nobody is in the room
```

```
peopleHappyIntervals : Integer derive:
    self.user->select(u|
        u.minComfortableTemp<=self.optimalTempIntervals and
        u.maxComfortableTemp>=self.optimalTempIntervals)->size()

peopleHappyWAverage : Integer derive:
    self.user->select(u|
        u.minComfortableTemp<=self.optimalTempWAverages and
        u.maxComfortableTemp>=self.optimalTempWAverages)->size()

peopleHappy : Integer derive:
    self.user->select(u|
        u.minComfortableTemp<=self.currentTemp and
        u.maxComfortableTemp>=self.currentTemp)->size()
```

A.2 Measurement and Occurrence uncertainties

The algorithms need to slightly change when occurrence uncertainty is considered, in order to have into account the confidence of each digital avatar. Changes with respect to their previous versions are marked in blue color **Bertoa** ▶ *No he podido ponerlo en blue color* ◀.

```
optimalTempIntervals : Real derive =
  if self.user->isEmpty then  -- no users in the room...
    self.standByTemp
  else
    let minmin : Integer = self.user->select(confidence>=0.05)->
        collect(minComfortableTemp.value())->min().floor*10 in
    let maxmax : Integer = self.user->select(confidence>=0.05)->
        collect(maxComfortableTemp.value())->max().ceil*10 in
    let maxUsers : Integer = Sequence{minmin..maxmax}->
        iterate(i; x:Sequence(Integer)=Sequence{}|x->append(self.
            ↪user->
            select(u|u.confidence>=0.05 and
                    (u.minComfortableTemp<=UReal(i/10,0.1) and
                     u.maxComfortableTemp>=UReal(i/10,0.1)
                    ).confidence()>=0.95)->size()))->max() in
        UReal((Sequence{minmin..maxmax}->select(i|
            self.user->select(u|
                u.confidence>=0.05 and
                (u.minComfortableTemp<=UReal(i/10,0.1) and
                 u.maxComfortableTemp>=UReal(i/10,0.1)
                ).confidence()>=0.95)->size()=maxUsers)->min()
                    ↪/10),0.1)
  endif
```

The second algorithm (WAverage) calculates a weighed average using as values the averages of the comfort ranges of each user, and the weights are calculated in inverse proportion to the square of the distance to the AC controller. In this case, more changes are needed because the confidence assigned to each user already have into account the inverse proportion of the squared distance to the AC controller.

```
optimalTempWAverages : UReal derive =
    if self.user->isEmpty then  -- no users in the room ...
        self.standByTemp
    else
        let weight : Real = self.user->collect(confidence)->sum()
            ↪in
```

```
            UReal((self.user->collect(u|
                ((u.minComfortableTemp+u.maxComfortableTemp)/2)*
                  u.confidence/weight)->sum()).value(),0.1)
    endif
```

The derived values of the rest of the attributes of class AC are specified in OCL as before, apart from the addition of confidence constraints to filter out those users with a very low level of confidence:

```
optimalTemp : UReal derive = (optimalTempIntervals +
    ↪optimalTempWAverages)/2

standByTemp : UReal init: UReal(20.0,0.1) -- when nobody is in the
    ↪ room

peopleHappyIntervals : Integer derive:
    self.user->select(u|
        u.confidence>=0.05 and
        (u.minComfortableTemp<=self.optimalTempIntervals and
         u.maxComfortableTemp>=self.optimalTempIntervals).
             ↪confidence()>0.95)
         -> size()

peopleHappyWAverage : Integer derive:
    self.user->select(u|
        u.confidence>=0.05 and
        (u.minComfortableTemp<=self.optimalTempWAverages and
         u.maxComfortableTemp>=self.optimalTempWAverages).
             ↪confidence()>0.95)
         -> size()

peopleHappy : Integer derive:
    self.user->select(u|
        u.confidence>=0.05 and
        (u.minComfortableTemp<=self.currentTemp and
         u.maxComfortableTemp>=self.currentTemp).confidence()
             ↪>0.95)
         -> size()
```

Acknowledgements This work was supported by the Spanish Ministry of Economy and Competitiveness under projects TIN2014-52034-R and PGC2018-094905-B-I00.

References