POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Sharon: a Smart Home Environment Simulator

Tesi di Laurea Magistrale in
Computer science and Engineering - Ingegneria informatica

Author: **Francesco Maria Di Rienzo**

Student ID: 944548
Advisor: Prof. Sara Comai
Co-advisors: Fabio Salice, Andrea Masciadri
Academic Year: 2022-2023

Abstract

The average age of the world population is increasing; in particular, it is forecast that the percentage of elderly people will grow as well. This fact will pose important challenges to the healthcare system which would be in charge of treating a larger number of people. To limit the pressure on the public healthcare system and help aged people live a better life in their own homes, it is possible to use a smart home environment that assists them in their ordinary lives . The problem is that to design a smart home environment, a lot of data is needed, both for the development of the hardware and the software. However, data is scarce because the simplest approach of gathering them using a real prototype of a smart home environment would not work. It would not work because it would be too burdensome in terms of cost and time to build a specific smart home environment for a specific place without a priori information; moreover, this installation would not be able to generate sufficiently general datasets. We developed a smart home environment simulator called Sharon to solve this issue. This simulator can produce datasets that can be used to build real smart home environments. The simulator can produce two datasets; one contains the scheduling of activities of daily living (ADL); the other contains the sensors' activations representing the status of each sensor in the virtual environment. The dataset containing sensors' activations is computed starting from the dataset composed of ADLs. In this thesis, we developed additional features to increase the software simulator's capabilities. The first feature that we added to the program is the fact that the user can force a set of ADLs in precise time instants at run-time, affecting the output datasets. The second functionality that we implemented is the ability to reproduce anomalous events in the form of a sequence of sensors' activations arbitrarily decided by the user itself. Examples of anomalous events can be the fall of the inhabitant or the fact that the dweller does not remember to close the fridge. The third feature that was coded is the ability to use Sharon to do real-time software testing of a real smart home environment. In particular, the user can choose the speed of the simulation and so the output datasets can be processed by a real smart home environment in real-time while they are being computed. The first two goals have been completely reached; the third one works if the translation of ADL scheduling in sensors' activations is based on a deterministic approach; instead,

when using a stochastic approach it does not work.

Keywords: Smart home, Human environment simulation, aging population, elderly care

Abstract in lingua italiana

L'età media della popolazione mondiale sta aumentando, in particolare è previsto che anche la percentuale di anziani crescerà. Questo fatto pone importanti sfide al sistema sanitario che sarebbe in dovere di trattare un maggior numero di persone. Per limitare la pressione sul sistema sanitario pubblico e aiutare gli anziani a vivere una vita migliore presso le loro abitazioni, è possibile usare un ambiente domestico intelligente per assisterli nella vita di tutti i giorni. Il problema è che per progettare un ambiente domestico intelligente è necessaria una grande quantità di dati, sia per sviluppare il software che l'hardware. Tuttavia c'è una penuria di dati poiché il più semplice approccio di raccoglierli usando un prototipo di ambiente domestico intelligente non funzionerebbe. Non funzionerebbe perché sarebbe troppo oneroso in termini di denaro e tempo costruire uno specifico ambiente domestico intelligente per uno specifico luogo senza informazioni a priori, inoltre questa installazione non sarebbe in grado di generare insiemi di dati abbastanza generali. Per risolvere questo problema abbiamo sviluppato un simulatore di ambiente domestico intelligente chiamato Sharon. Il simulatore è capace di produrre insiemi di dati che possono essere usati per progettare ambienti domestici intelligenti reali. Il simulatore può generare due insiemi di dati, uno contenente la programmazione delle attività giornaliere (ADL), l'altro comprendente le attivazioni dei sensori, che rappresentano lo stato dei sensori presenti nell'ambiente. L'insieme di dati che contiene le attivazioni dei sensori viene calcolato a partire dall'insieme di dati che contiene le ADL. In questa tesi abbiamo sviluppato funzionalità addizionali al fine di aumentare le capacità del programma di simulazione. La prima funzione che abbiamo aggiunto al programma è il fatto che l'utente può forzare durante l'esecuzione un insieme di ADL, in un preciso istante temporale, che ha un effetto sugli insiemi di dati prodotti. La seconda funzionalità che abbiamo implementato è l'abilità di riprodurre eventi anomali nella forma di sequenze di attivazioni di sensori arbitrariamente decise dall'utente stesso. Esempi di eventi anomali possono essere la caduta dell'abitante o il fatto che il residente non si ricorda di chiudere il frigorifero. La terza funzione che abbiamo codificato è l'abilità di usare Sharon per fare un collaudo di un ambiente domestico intelligente in tempo reale. In particolare l'utente può scegliere la velocità di simulazione e quindi gli insiemi di dati prodotti, mentre sono computati,

possono essere processati da un ambiente domestico intelligente in tempo reale. I primi due obiettivi sono stati completamente raggiunti, il terzo funziona se la traduzione da programmazione di attività in attivazioni di sensori si basa su un approccio deterministico, mentre quando si usa il metodo stocastico non funziona.

Parole chiave: Casa domotica, simulazione di ambiente umano, invechiamento della popolazione, cura degli anziani

Contents

Introduction

According to various reports [3, 5, 16] among which the one from the United Nations [4], by the middle of the century, one-sixth of the world population will be at least 65 years old, up from one-eleventh in 2019. These reports also suggest that the world is undergoing a "longevity revolution" according to which the amount of life that is spent above age 65 is raising considerably and this means that there are a lot more people reaching old age. According to the World Health Organization [2], by the middle of the century, the number of people older than 60 will nearly double for 2015. Another important argument is that, by 2050, 80% of old people will be living in low and middle-income countries. In the article published by the International Monetary Fund [23] it is pointed out that the aging of the population will have an economic impact, in particular working people will pay more to support the elderly and also a bigger part of the public spending should be allocated to support health and retirement programs. From these forecasts, the problem of how can low- and middle-income deal with the aging of their population arises. If we analyze the problem more deeply we can notice that there is also a time constraint all the forecasts will occur by 2050 so all the countermeasures should be ready for that time. In this master thesis, we will try to find a solution to this urgent problem respecting the temporal and pecuniary constraints. This problem is relevant because it is common in all countries. After all, the tendency to live longer is spread all over the world. Not only this problem will be extremely diffused as already said, but it will have a significant impact on the life of people, specifically, those people that are in the working age and that will be obliged to pay to sustain the health of the elderly. There are already costly ways to deal with this problem, but precisely these are expensive and we want affordable solutions for low or middle-income countries and people. One of these expensive ways is hiring a caregiver to assist the old person in his/her own home, but this method is not always possible because not all families have the economic possibilities to hire a cohabitant caregiver, moreover, this operation, has the effect to make the aged adults feel old and this often leads to refuse of help, this intervention is highly invasive from the perspective of an old adult who will lose part of his/her freedom and privacy. Another method to take care of the elderly without weighting on the state budget is to entrust the old adult to a retirement

home, but also this solution is quite expensive and presents various problems related to the fact that the person will have to stay with other unknown persons and will have to live in a completely different environment that will deeply affect its freedom and its habits. Moreover, both the two discussed solutions, have a common problem: they rely on the fact that the elderly would be physically assisted in doing activities of daily living by a caregiver who is not part of the family unit. Nowadays this approach is dangerous because of the spread of the Covid19. Consequently, a solution that reduces contact between people would be preferred because less dangerous. From these considerations and problems, the idea was to create a smart home that would be able to monitor the elderly in his/her own home, allowing him/her to keep living a normal life without too much intrusion, and to notify the relatives of the dweller only in case of anomalies. This approach would solve the economic problem because a smart home is less expensive than hiring a cohabitant caregiver or entrusting the old adult to a retirement home. Also, the issue of the Covid19 diffusion is decreased because the inhabitant's contacts with other people external to the family are limited. A person would be more willing to accept a set of electronic devices that constitute the smart home environment than a new unknown cohabitant person or a new environment which is the retirement home, so we can state that there is a higher probability that this solution would be accepted by the elderly. The work described in [9] focuses on the acceptance of the Internet of Things environment, a network of sensors, actuators, digital devices connected through the Internet, from the inhabitants, especially this study shows that compatibility, ease of use and usefulness are the among the best indicators of the intention to use the IoT system, also cost has a significant impact. In other words, we are proposing a known approach called Active and Assisted Living that should avoid the hospitalization of the elderly ensuring a high quality of living to the old adult and a high-security standard regarding the elderly status to the family [1, 24]. The smart home environment should retrace the IoT architecture where the digital devices, sensors, or actuators, in a home, are connected to a central server that can compute the information coming from the home of the elderly and should be able to provide the relatives with a detailed report of the status of the monitored old people. To do so the server should use Artificial Intelligence techniques and algorithms, but these algorithms rely on a model that has a variable number of parameters, the more the parameters the more the model is complex, that needs to be estimated, and to estimate them you need training data, and the more is complex and powerful the model, the more the data that you need. Moreover to validate and test the model further data are needed, that is to say, that these approaches are data eager. Unfortunately, several reasons determine a shortage of datasets with a good quality that can be used to train these smart home environments servers that should compute the Artificial Intelligence

algorithms. For example, if you wanted to put in place a smart home environment that collects data in a real home, with a dweller or more, you should recruit that dweller, you should design and built the system, both the hardware and the software, also you would need money investments for the design and the production phase. In this scenario, it is important to consider that the data collection would be extremely slow because it should respect the pace of life of the people involved in the experiment. Furthermore, if there is a mistake in the design or production phase, and you can discover that only after that you have already gathered some data, it would be necessary to do again the data collection, again at the same pace, the iteration of these two phases could cause a remarkable waste of time and money. Even after having designed the smart-home environment that should be used to gather the data, it is not given that it will work according to the requirements so it is important to verify that it is working properly. To be sure that the system meets the requirements the people living in the environment must write down what are they doing together with a description of the context in which the data was collected, this additional information is called "ground truth". This "ground truth" however is not reliable because people often forget to compile it. The specific problem that we want to solve with this thesis is to find a different way to gather data that will be used to design a smart home environment for helping the elderly and their families, in a faster and cheaper way than proceeding with iterations of designing the initial smart home, producing it, putting it in place, collecting data, checking the results and repeating if needed.

Here at Politecnico di Milano, in particular at the Assistive Technology Group, (ATG), a project of this kind has been developing: a wireless sensor network that can perceive the activities of the people in the environment in which is installed, and can send notifications to a visualization tool that will provide all the needed information to the relatives of the old person. The architecture of the whole system is one of the Behavior dRift compensation for autonomous and InDependent livinG, (BRIDGe), project. BRIDGe is the visualization tool that receives the information from the smart home environment and computes them to give aggregate visual information and reports. To produce the data that is needed to train, validate and test the Artificial Intelligence algorithms and techniques that are used in BRIDGe and also to test other functionalities of that project, the ATG has been developing SHARON which is a human behavior software simulator that can output text files representing the scheduling of Activities of Daily Living, ADL, over many days that can be chosen, and it can also output text files representing sensors' activations corresponding to that high-level scheduling. A more detailed explanation of how SHARON works and its functionalities will be given in the method section. The important message is that the SHARON software simulation allows us to generate data

in a controlled environment, in a very short time with respect to the time that would be required to gather the same amount of data in a real scenario, and in a much cheaper way than designing and building an actual smart home capable of collecting data. In the subsequent chapters, we will illustrate the state of the art, the method, the results, and the conclusions.

1 | State of the art

In this chapter, we will present, analyze and classify the relevant literature regarding the smart home environment simulators and useful tools that have an impact on the functions of this kind of software. In particular, we wanted to find significant functionalities implemented in other simulators to enrich Sharon with them. Another important purpose of this inquiry is to be able to classify Sharon with respect to the current state of the art.

1.1. Persim

Persim is a software [11] that is able to simulate, human activities. It is event-driven, it can generate datasets in a standard format which is the Sensory Dataset Description Language, SDDL, in order to facilitate the sharing of the data. It can be configured to capture micro or macro events. Its use cases are:

- Converting a dataset into SDDL.
- Simulate human behaviour in a smart environment and generate the corresponding dataset.
- Share datasets through a repository.
- Extend a given dataset by modifying the environment in which it was computed.
- Validate a general dataset by comparing it against a real one.

There are five steps that must be done in order to be able to simulate a smart home environment. You have to create the space and areas, then you can choose among a set of sensors and actuators and place them in the space, then you express the activities to be executed, then you define the mapping between activities and sensors that specifies which are the sensors relevant for each activity, and in the end, you can start the simulation. The user can set all these parameters through a graphical user interface. There are two modes: Activity-Driven and State-space. In the Activity-driven mode, the user creates the list of activities and the mapping between sensors and activities and the simulator produces the dataset. In State-space mode a series of events, fired at a certain timestamp,

is created and for each of them the proper activation of all sensors is generated, that is a sensor can be activated or not by a certain event. It is not possible to model outdoor activities.

1.2. DiaSim

DiaSim [8] is a software that can simulate a smart home environment, it has parameters that allow modifying the target environment according to the needs of the user. The set of parameters describes the environment which should mimic the real environment. Then it is possible to execute the applications in this environment and a platform that allows the development of simulation logic which can be tested in the former environment, a special debugger is also integrated.

1.3. Simulation of a smart home environment

PlaceMaker is a utility software that can be used to generate a matrix that can have two or three dimensions and that represents an environment, and a connectivity matrix, which represents paths between rooms, is associated with the matrix representing the environment. V-PlaceLab is software that is used to create a list of activities that should be executed by the virtual agent. VPlaceSims is software that gives the user the ability to move the avatar in a graphical environment and allows the virtual agent to interact with others and also with objects. Now we present this peculiar work [6]. A utility to draw the house map is presented, it gives the user the ability to draw on a canvas the various elements such as walls furnishings, each of these entities has its properties such as size that can be modified. It is also possible to insert sensors: PIR sensors detect human presence using infrared, PM sensors detect human presence using pressure. Each home space that is created with that tool can be saved in XML format, and it is possible to load an XML file representing a floor plan. It is possible to define an agent profile and also a list of activities for each agent, each activity can generate a set of signals which are the output of the sensors in the environment. The simulation can be repeated to increase the size of the data collected which can be saved into a database. As a recap, the input for the simulation is the structure of the home, the resident profile, and the schedule. The simulator enables the user to see the virtual dweller moving in the virtual environment during the simulation.

1.4. IE Sim

IE Sim is a software simulator [15], endowed with a graphical interface, that allows the user to create a dataset composed of ADLs deriving from the activity of a dweller in a smart digital environment. In the simulator, there is a utility that allows the user to create a 2D plan, with objects and sensors which have properties that can be modified. The user moves the virtual agent in the virtual environment using the keyboard, the virtual agent can interact with the sensors in two ways: a passive interaction or active interaction in which the user has to use the interaction menu to interact with them. The generated dataset is in the homeML standard.

1.5. Simulating Events to Generate Synthetic Data for Pervasive Spaces

In this work [19], the authors present a model that will be used to carry out the simulation, that is based on Markov chains, probability distributions, and Poisson processes. According to the work described here, Markov chains are indicated as useful to produce patterns of activities given a priori information about the target schedule. To assign timestamps in a given time interval it is suggested to use Poisson distribution. The generation of the activations of the sensors is based on probabilities distributions that are tuned according to the characteristics of the sensors.

1.6. Multilevel Simulation of Daily Activities:Why and How?

This work [13] presents a very interesting state of the art section in which it is pointed out that it is possible to adopt a Markov model to estimate the location of the virtual agent. Another important reference is to Cardinaux where a model updates the probability that the dweller will perform a certain action according to how much time has elapsed from the previous execution of that action, after each execution that probability is set to zero. The virtual dweller is represented using a set of parameters, a part of them represents the character of the person such as how many hours of sleep a person needs, the remaining part represents the needs of the person such as tiredness. Given this set of variables, which represents the current status of the person, the next action that has to be executed can be determined. There is also the description of the virtual environment that consists of a plan of all the locations where the dweller can go, also in this case there is a set

of parameters associated with the virtual environment, such as how many doors there are. In the environment both motion and non-motion sensors. There are three types of activities: high-level activities, middle-level activities, and low-level activities, so there is a hierarchy between them. A certain top-level activity is scheduled if its corresponding priority function is the maximum among the set of priority functions, this is a sort of schedule according to the needs of the person, if the person is hungry then it will eat. The middle-level activity is more fine-grained than a high level one in the sense that a high-level activity is composed of a sequence of middle-level activities, for instance, the high level can be clean the bathroom and the sequence of the middle level can be clear the bathtub, clear the sink and clear the floor. The state of the simulation and the output is produced when the bottom-level actions are performed, in particular, these actions will influence the variables and the parameters that are used to describe the human character and the environment. The output of the simulator consists of a set of text files containing the logs of the various sensors, also logs that contain high and low-level activities.

1.7. Computer simulation of the activity of the elderly person living independently in a Health Smart Home

An important discussion here [20] reported is about the fact that the data that comes out of the sensors and get stored as a time series is incomplete, two reasons are mentioned: the sensor can fail and produce false positive or false negative detections, the second one is that there is a lack of people that are willing to stay in the monitored environment to trigger the sensors and produce the data. The data is preprocessed, re-sampling it at 1 Hz, subtracting the median, and passing over a first-order hold filter. The next step is the clustering of the data using Data Mining techniques, then this data is used as transition elements to compute two Markov matrices of transition. Then, to start the simulation, the two previously computed matrices are used as input for a Matlab function called HMMGENERATE that will give as output the states and the sequence. It is presented experimental data gathered over 1492 days from a real smart home environment where passive presence infrared sensors were used. The collected data is organized as a time series where there is the timestamp, the date, and the localization of the dweller.

1.8. Simulation of home daily activities for lifestyle monitoring systems development

In [10] the authors present a software simulator that is based on a hierarchy of two layers, the first and the higher one represents the schedule of the activities that the dweller does in the environment, the lower layer represents how these actions take place, how they are carried out, an activity is composed of a series of actions, actions have not a duration time, activities duration time is determined by the actions that compose it. The environment is coded as a graph, edges are locations and transitions are vertices. The route path of a moving agent is computed using the shortest path. This is a parameterized software for example there is a parameter that regulates the moving speed of the agent. The parameters can change at run time according to various factors such as the health of the simulated agent. The scheduling of the activities during a day depends on the needs of the person which vary over time, in particular, a need increases until it's satisfying. A Bernoulli trial will decide at each iteration if a certain activity that has a certain probability, depending on the urgency of the related need, to be executed has to be executed. The probability of performing a certain action grows linearly with the elapsed time and a constant, so that constant is a parameter of the system, in fact, it directly influences the interval of time between one instance of action and another one. In this simulator there are a lot of parameters, they are divided into four classes: Subject parameters, sensor parameters, change parameters, and environmental parameters. The subject parameters are the ones that influence the subject conduct. The sensor parameters regulate the detection of events from the actions of the agent. Change parameters are deputed to represent possible changes in the needs of the agent. Environmental parameters describe entities such as weather, television programs, and so on.

1.9. CASS

An important contribution presented in [21] is the definition of context, which can be of two types, one is called primitive context information and the other is called complex context information. The first one represents the data coming from the sensors enriched with the time and location of the sensor that gathered the data. The second represents information that contains knowledge that derives from primitive context information. Another concept that is introduced is that of contextual rule, which is composed of a condition and an action, the condition can be expressed in terms of primitive or complex context information, the action part consists of actions that can be done by devices and

actuators. The software includes a graphical interface that helps the user in defining the contextual rules, it can also understand the conflicts between them and displays the control movement of the digital agent. The software can produce its primitive context information to check and test the rules. This simulator has three kinds of virtual agents: the administrator, the authenticated user, and the guests, the three types have different privileges, for example, the guest cannot control the smart home. The software allows the user to set up the virtual environment, containing the sensors and actuators, in which the simulation will take place.

1.10. SIMACT

This simulator [14] can set up, with a visual tool, a three-dimensional environment. An interesting feature is that a plan is defined as a series of steps, but it can also contain steps that are not useful for reaching the goal of that plan; it is possible to see, thanks to a graphical interface, the execution of the plan and its steps. The simulation speed can be set according to the user's needs. The sensor's data can be saved into a database so that this data is available to other applications. The tool works out of the box thanks to a set of scripts, but the user can create new scripts to personalize the simulation, where each script describes a series of actions in the virtual environment. It is clear that in this project the user interface is very advanced. A choice that also they have made is to use SketchUp for the creation of the 3D environment. In this approach, the activity model is a composed one, in the sense that there is a hierarchy where activity is composed of a set of actions.

1.11. A Simulator for Generating and Visualizing Sensor Data for Ambient Intelligence Environments

The smart home environment simulation software described by Buchmayra, Kurschlb, and Küng in [17] can have as an environment a two-dimensional floor plan where the user can locate the sensors that can be also defined by the user. The sensor model can represent the fault of the sensor or the corruption of the data caused by the noise by altering the output of the sensor itself or by activating a sensor that should not be activated. The output data is available to external components in real-time.

1.12. Development of a smart home simulator for use as a heuristic tool for management of sensor distribution

The work presented in [22] describes a software simulator that is based on a 3D virtual environment where the virtual dweller is moved by the real user through the keyboard of the computer, the virtual agent will interact with the sensors placed in the 3D environment and the actual user can see the simulation in the 3D environment from different perspectives such as a first person or third person. The objective of this simulator is to help the researcher to identify the best sensors placement given a certain environment.

1.13. Simulation of smart environments

The smart home environment presented in the work of Armac, Ibrahim, and Retkowitz [7] is called eHomeSimulator. The environment is represented in a peculiar three-layered way: there is the actual floor plant that describes the position of the various rooms, there is the virtual agent, and there are the sensors. The display is in 2D, and the three layers are: the lower one is in charge of representing the background, the middle one must represent the sensors in the environment, and the higher one is devoted to the representation of the digital dweller in the smart space. Various external tools are used to render the environment, for example, Google Sketchup for the background image. An environment editor is provided within the software which allows to define and create accessible and not accessible areas and to locate sensors in the environment. The graphical user interface (GUI) of the simulator allows the user to select the virtual agent that will be moved, there can be more than one dweller but only one at a time can be controlled by the user. Also, the sensors' output can be viewed from the GUI.

1.14. Classification and analysis of literature

The classification of the software that is used to simulate the smart home environment, which is presented in [12] is very significant. In particular, the classification is based on how the output is computed and does not take into account additional features that do not contribute to the creation of the output, for instance, a nice feature that is does not determine the category of software is the possibility to have more people in the same environment, another feature that does not influence the classification is the ability to define new sensors. The classification that is presented supposes two classes: the

model-based class and the interactive-based class. If the software is model-based then it means that there is a mathematical model that defines the succession of activities and the duration of each activity. We have found in the literature that for these objectives the hidden Markov model is used to compute the schedule of the activities and the Poisson probability distribution is used to calculate the duration of an activity. An example of model-based software is SIMACT and Persim. The software that is based on an interactive approach produces its output thanks to the direct interaction of the real human user that defines in some ways, mouse or keyboard, for example, the movements and the actions of the virtual agent in the simulated environment, so the key part of this family of software is the GUI that allows the interplay. Also in this kind of software, there could be some code representing a mathematical model, for example, the activation of a sensor is not represented in a deterministic way but in a stochastic way using some probability distribution. An example of the previously analyzed software that belongs to this class is the CASS simulator. From this classification, interesting observations arise. The model-based approach produces a dataset that derives from the models that it uses, so the quality of the dataset depends on the quality of the model. The model is composed of parameters and hyperparameters that need to be estimated from real data, but as we already said, in this field, there is a shortage of real data. Moreover, these models need to be complex to have the sufficient power to describe human actions in a smart environment and so they must be complex. This fact brings another disadvantage: it is difficult to fine-tune the model and to reconstruct the computation of the results. The advantage of this approach is that it can generate a dataset over an extended period of simulated time in a short real-time. Regarding the advantages and the disadvantages of the interactive approaches it is clear that the user has fine-grained control over the activities that the virtual agent is going to perform in the virtual environment, this allows a better explainability of the results of the simulation. Another advantage is that a model of the behavior of the dweller is not needed, this is important because as we have already said, to build its data are needed and in this area, they are few, instead of in this interactive approach the complexity shifts to the design of the virtual environment and the GUI of the simulator.

1.15. Literature review

In this chapter, we will try to comment and grasp the most interesting and useful results that we have found in our research. In particular, we will point out some features that could be integrated with Sharon to make it better.

The first comment that we make, addresses an important problem of the field: there is

not an established standard dataset representation used by the majority of the simulators. This is important for many reasons, first of all, it slows down the advancement of the knowledge of the field because it makes comparisons between simulators more difficult. Another reason is that it would be very useful to be able to combine datasets coming from different simulators to get a much larger dataset, that is also considerably more general and so more qualitative. We think that having such a dataset would be a dream scenario because the machine learning algorithms would get a significant benefit from the greater quantity and quality of the data. We think that there is already a feeling in the field for this problem, as evidence of this, there are various projects that provide utilities that can convert a dataset in one format into a dataset in another format such as Persim does. The dataset translation is a feature that some simulators offer, we have found that this is not the only functionality that is often provided. Some simulators offer functionalities that can be used to produce configuration files that will be used by the simulator itself, for example, a common utility is the one that allows the user to design and build the map of the environment that is usually coded as a graph. Another common utility is the one that allows the user to create a list of ADLs representing the scheduling of the activities of the virtual agent and this high-level scheduling is used to produce the sensors' activations. Some other simulators make extensive use of external tools such as SketchUp for the creation of the environment, PlaceMaker that allows the creation of matrices representing the environment, and VPlaceSim that allows the user to see the avatar moving in the environment. Both these two types of utilities are important because the real challenging part of a simulator is how is the high-level scheduling of activities produced and how it is mapped into sensors' activations and we think that it is important to direct the resources to develop these two points rather than investing them on secondary features that can be reused. In other words, we are suggesting for someone willing to make a human environment simulator to take the utilities that are already developed and tested and concentrate on the models of the dweller and the sensors. Another pattern that we have detected is the use of a GUI, in particular for those simulators that need the intervention of the user to move the dweller in the virtual environment and generate the sensors' activations. For what concerns the simulators that generate the high-level scheduling of the activities and the low-level dataset with a mathematical model, commonly, the designers of those algorithms organize the code to reflect this hierarchy, several projects assert to have two or even three layers, the higher one is in charge of computing the high-level scheduling and the lower one has to duty to translate that in sensors' activations. This feature is detectable also in Sharon where the high-level layer computes the list of ADLs and the lower layer the sensors' activations. In particular, these kinds of simulators often use the Markov model to produce the high-level scheduling and the Poisson probability

distribution to decide how much a certain ADL should last, also the model of the sensor is usually probabilistic. Sharon uses in the computation of the high-level part both Poisson probability distribution and Markov models and also the model of the sensor is stochastic. The advantages and disadvantages of using a model-based simulator have been already discussed before. Another feature that is often implemented in these simulators is the possibility to save the produced datasets in databases. This feature could become crucial if the produced datasets become too large or too complex to be handled only as text files because managing them would be too burdensome with respect to the execution time. Among the described ones we think that two features would improve the usability of Sharon. The first one is adopting the most used format for the sensors' activations dataset and handling it with a database. The second one is providing the user with a tool that may be integrated into the simulator, which makes the process for creating a new representation of the virtual environment simpler than before.

2 | Method

In the first part of this section we will illustrate the functioning of Sharon, then we will present the changes that we have done following an approach that consists in describing the goal that we want to reach and then describing how to actually change the source code of the program to accomplish it.

2.1. Sharon framework

According to the classification provided in the state of the art section, Sharon is a human environment software simulation, based on a model approach, so there is a model and parameters that govern the computation of the output. The model, is based on a set of needs that involves both the virtual dweller and the house, in particular, these needs are:

- Tiredness: how much a person needs to rest.
- Boredom: how much a person needs to perform some entertaining activity.
- Hunger: how much a person needs to satisfy its appetite.
- Stress: how much a person needs to relax.
- Loneliness: how much a person needs to interact with other people.
- Uncleanness: how much a person desires to wash himself/herself.
- Excretion: how much a person needs to defecate and urinate.

There are also needs that are linked to the environment and not to the dweller:

- Low Stock: it represents how few things are in the house.
- Untidiness: it represents how dirty the house is.

For each of these needs, there is a variable σ_n that represents the urgency of the related need to be performed, it takes decimal values between zero and one. These variables representing needs evolve through time following a linear model, in particular, to explain this model it is necessary to introduce further notions. Each need has a spontaneous

growth rate γ_n that represents how much that need is increased from one time instant to another, for example, it is clear that during the day a person gets constantly tired over time, the same reasoning can be applied to the other needs. Another concept that needs to be introduced, is the effect of specific action on the urgency of a certain need ζ_n, there is a quantity that can be positive or negative and that is a function of the action that has been performed that influences the future value of the variable representing the urgency of a specific need, for example, if the virtual agent has eaten than the effect of the ongoing activity would be a negative number, that would decrease the variable representing the hunger. The final equation that represents the status of a need as a function of the time is the subsequent one.

$$\sigma(t)_n = \sigma_n(t-1) + \gamma_n + \zeta_n$$

Another function that is necessary to be defined is a scoring function that takes in input a certain action and the set of variables, each representing the urgency of a specific need, and returns as output a decimal value between zero and one that describes how much that specific action is good taking into account all the needs that the virtual dweller has in that specific time instant.

Scoring function's definition: $N(a, \sigma_n(t)) = \sum_n w_{n,a} * \sigma_n(t)$

In particular, this scoring function computes the weighted sum of the set of variables representing the urgency of each need. The weights $w_{n,a}$ involved in the weighted sum represent how much a certain need n is satisfied by that action a. This scoring function is not sufficient to be used to produce a schedule of activities for the virtual agent because it does not take into consideration the periodicity of actions of a human being it is well known that most humans will have lunch around noon and they will have lunch after seven post meridiem. There is a function $\theta_a(t)$ representing time dependency for each possible action that takes in input time and returns as output the probability to execute that specific action in that specific time, so it is a decimal value between zero and one. According to this function, there would be a deterministic series of actions, but it is necessary to add a random element to obtain a more realistic model, in particular, a random variable υ that can take values between zero and one is extracted, if $\upsilon > \theta_a(t)$ then the actual likelihood function would be $\Theta_a(t) = \theta_a(t)$ otherwise $\Theta_a(t) = 1$. The true final score function that is computed for each action and takes in input the current time, returns a value between zero and one, in particular, it is the product of three terms: the first one is a factor that discourages the activity change, so if the current activity is the activity for which the score is being calculated, then its value is one otherwise it is a constant less than one, the second factor is the scoring function defined before that takes

into account the set of needs, and the third component is the timing function with the random heuristic.

Final scoring function's definition: $S(a,t) = \alpha(a) * \sum_n N(a, \sigma_n(t)) * \Theta(a,t)$

As already said, Sharon is a smart home environment simulator that is based on a model which by definition has a set of parameters that characterize it. In this case, the parameters used by the model present in Sharon are:

- The set of ADLs that the virtual agent can do in the environment.
- The set of needs of the dweller and the set of needs of the house.
- The set of $w_{n,a}$ which represents how good a certain action is to fulfill a certain need.
- The set of spontaneous growth rates σ_n each of which represents how much a certain need increases from a timestamp to another.
- The set of effects of each activity over each need γ_n.
- The set of functions, each for each action, representing the likelihood to perform it in a specific timestamp $\theta_a(t)$.

These parameters have been obtained in two ways:

Interview analysis is a method that consists in directly asking an interviewed person the activities that he or she performs during the day, what are the needs that these activities satisfy, and what are the times during the day in which they are performed. The data deriving from these interviews have to be conscientiously taken into account because it strongly depends on the person that has provided them and so there is a bias.

Real dataset analysis and diary analysis, the first approach consists in collecting data from a real smart home environment installation, the second one consists in asking a real dweller to fill in a diary in which they report the scheduling of their activities for each day and the duration of each activity. From this type of approach, it is possible to gain quantitative data.

Until now the generation of the high-level scheduling of the actions has been described, but Sharon can generate, starting from the high-level scheduling, the sensors' activations which are the output that would be produced if these sensors would be in a real smart home environment monitoring the actions of a dweller that does the actions contained in the high-level scheduling. There are two ways in which Sharon can produce this output from the high-level scheduled list of activities that it has already produced: the first one is

based on the behavior of the virtual dweller, the second one utilizes a stochastic approach to produce the synthetic output data. The Sharon simulator is thus able to both produce new datasets and extend one already existent. It is also possible to import the high-level scheduling and generate using it the low-level sensors' activations.

Now the model of the agent in the software will be discussed. In Sharon, the agent has two properties: its location, described by a couple of coordinates, and a maximum walking speed that represents the maximum distance that the agent can cover between two-time instants. It does not accomplish any action and in fact, it triggers the activation of a sensor only according to their relative position, in other words, it is just the presence of the agent that activates a sensor.

Now the virtual environment model will be discussed. The plant of the building in which the simulation takes place is discrete and it contains walls, doors, and furniture, it is composed of tiles that describe the possible paths that the agent can run across, in particular, the agent will follow the shortest path for going from a position to another. In particular, the environment is modeled using a matrix of zeros and ones. The model of the sensor has three properties: the position of itself, the area in which a passing agent can be detected, and the probability distribution that is the probability of success of a Bernoulli trial used to determine the output, all of the sensors have a binary output.

The computation of the low-level dataset containing sensors' activations can be carried out in two different ways: a deterministic manner and a stochastic manner. We will now discuss the deterministic manner. Each high-level activity that is scheduled is used to produce a further output representing the sensors' activations deriving from the high-level scheduling, but it is common knowledge that each activity, even of the same time duration, can be carried out in different ways, for example, if a dweller cooks pasta then the sensors' activations of the kitchen will be different then the case in which the dweller decides to cook fish and chips. Thus a single high-level activity can be represented at a lower level in different ways. This can be represented and obtained by associating a set of patterns to each activity. A pattern is a sequence of tuples of two elements, one represents a position and the other represents the duration for which the agent stays in that position. The execution of a pattern is carried out in this way: the agent moves to the first position that is in the first tuple and stays in that position for the time that is in the same tuple, then the second tuple is processed in this way and so on. It is important to notice that the agent will follow the sequence of positions in a deterministic way. During the walking from one place to another, the virtual agent will activate the sensors that will produce the lower output dataset. The creation of the various pattern can be done without considering real data but only using common knowledge, for instance, the activity "Watching TV"

can have as a lower-level pattern a sequence of tuples where the virtual agent spend the time sitting on the sofa or another possible pattern of the same activity can be one in which the agent sits on a chair for a different time.

Now we will discuss the stochastic approach. There are several ways to act, in fact, the generic activity "washing" can be done in various ways for example "washing a t-shirt" is not the same as "washing the clothes for the family", in particular, these two specific activities differ in the number and type of the subtasks. To cover this difference it is necessary to introduce the concept of Activity Template which specifies in which way a certain activity should be performed. It is also required to specify the concept of sensorset (SS) which is a vector of size N, where N is the number of sensors in the virtual environment, that has at each position i the output of the i-th sensor. An Activity Template is a tuple characterized by four elements: the probability to be executed given the actual high-level activity ("washing" in the previous example), the probability of a certain SS to be the initial SS, a transition probability matrix between SSs and the change rate between SSs. In particular, the sensorset changes can be described with a stochastic system with finite state space where each state is a sensorset and the general framework that can is used to describe our solution is the Semi-Markov model [18, 25].

The model of the sensor is not deterministic but stochastic in the sense that its output will not be one, that is the sensor is active, just because the agent is in the area that the sensor can monitor, but the fact that the agent is in the area is a necessary precondition to get the result of a stochastic process that will determine the output of the sensor, the probability distribution, on which the process is based, has been determined with real data.

2.2. Important implementation details

In this section important information about the implementation and the usage of the Sharon simulator will be given.

To be able to use the simulator it is necessary to download the source code and it is also necessary to install Ant, which is a java library and a command-line tool. The code for the public will be made public at a later time. After downloading the source code and after installing Ant, to start the simulation it is necessary to open the terminal and navigate to the folder of the source code, at this point the user has to give the command "ant" to start the simulation. At the end of the simulation, the high-level scheduling of activities will be under the data/ActivityOutput and in particular, in this folder, there will be as many folders as the simulated days, the lower level output representing the sensors' activations

can be found in the folder data/SensorOutput.

Now the necessary files for the configuration of the simulator will be presented. The configuration file about the needs can be found at the following path config/need.conf in the Sharon folder. Each row of this configuration file represents the configuration of a need which consists of the ID of the need, the name of the need, the spontaneous growth rate of the need, and the optimal initial value for that need, each of these values is a comma-separated. In the config folder, it is possible to find a .conf file for each ADL, the first row of the file says which is the activity that it represents, the second row contains the effects that the ADL has on each need, in the third row there is the name of the needs that can activate the ADL, each value in position i of the seventh line says which is the probability to execute that ADL in the i-th day, the ninth line indicates how that ADL is influenced by the weather, then there is the probability distribution over each minute of the day that represents how likely is to perform that activity in that specific minute of the day, at the end of the file it is reported the minimum duration of the activity. The previous part is mandatory for the execution of the simulator, the fact that the simulator can reproduce also the sensors' activations is a plus that can be turned off and so the configuration part regarding this part of the program is optional. In the folder, config/env can be found various .conf files regarding the low-level simulation. The map.conf file contains the plant of the house saved in a binary way, zero values represent the area in which the agent can walk, one values stand for walls, at the end of the file there is the scale of the representation expressed in points per meter. The sensors.conf file, which is in the same directory, contains the configuration for each sensor, in particular, each of that files represents a sensor in the form of a string where the information regarding that sensor is comma-separated. The first parameter of the row is the name of the furniture where the sensor is located, the second and the third identify the position of the sensor from the center of the map in centimeters. The next four parameters identify the rectangular area in which the agent can be detected and the final parameter identifies the probability that the sensor is triggered if the agent passes in the area. The area and the probability can be omitted and are inferred at run time with values of zero and one respectively. The places.conf file represents what are the furniture in the environment and their location, also in this case each row of the file matches an element. The sensorsId.conf file is just a file that contains a string composed of comma-separated id values each for every sensor in the environment. The sensorsets.conf file represents all the possible grouping of the sensors' Ids that can make sense. In the folder config/patterns, there are various .conf files each of which represents the execution of a pattern of activity. An important consideration that must be done is that Sharon does not provide any utility that can facilitate the creation

of the change of the various configuration files, this is a problem because it decreases the variability and generality of the results, for example, it is quite tricky to manually change the map.conf.

Now the format of the output of the simulation will be analyzed in greater detail. In the folder data/ActivityOutput will be generated as many .txt files as the days simulated and each name will be of the type "DAY_*" where the "*" is a number that denotes the simulated day that the file refers to. Every first line of each of these files is a sort of reminder in the sense that it explains everything of the subsequent lines, in particular, it is a string that contains a comma-separated list of strings where each of these strings is composed of one or more word in natural language, the whole string represents a pattern for the subsequent lines. The first pattern string is the following "simulation_seconds,activity_id,activity_name,time_of_the_day" and it means that the first value of each of the following lines will be the elapsed time from the start of the simulation, the second value will be the Id of the activity that has been scheduled and so on for the other parameters. In the folder data/SensorOutput there is the optional output which represents the sensors activations, in particular, each line of this input represent the state of all the sensors, the i-th number represents the state of the i-th sensor and can be either zero if it is inactive or one if it is active.

It is possible to change the context of the simulation. It is necessary to have downloaded the source code of Sharon, it is also advisable to have installed an IDE to open the project that is written in Java so also Java should be installed. The following reasoning applies even if the user decides to modify the source code using a simple text editor. After opening the chosen IDE it is necessary to open the Main.java class, here there are all the parameters that the user can fine-tune according to his/her needs and desires.

The parameters are:

- def_simulatedDays: integer variable that represents the number of days of simulation.
- ENABLE_SENSORS_ACTIVITY: boolean that if set to true allows the generation of the sensors' activations.
- PRINT_LOG: boolean that allows the user to choose to have the logs or not.
- DISABLE_PATH: boolean that disables the path during the simulation.
- USE_DRIFTS: boolean that activates the drifts that change the needs during the simulation.

- USE_HMM_LL: boolean that forces the software to generate the sensors' activations using the probabilistic model.
- GENERATE_HL_SCHEDULING: boolean that allows the generation of the high level scheduling if true, if it is false then the high level scheduling is imported from a configuration file.
- MIMIC_ARAS: boolean that if set to true will force the output to have the same format of the ARAS dataset.
- WALK_SPEED: it is a double that allows the user to change the walking speed of the agent.

2.3. My contribution to the work

In this section, we will go through the various features that we discussed and that we implemented. We will follow the temporal order of the challenges that we have faced and we will present them by enunciating the problem or the issues that were presented to us, the reasonings that we have done, and the solution that we have reached.

2.4. Utilize Sharon to do software testing

The first objective that was given to us was to modify Sharon to be able to use it to software test another software that in this particular case is BRIDGe. BRIDGe is a software that can receive the information from a real smart home environment installation and send reports and notifications to the relatives of the dweller, so they can check the health status of the inhabitant. It can also send alerts in case a bad event happens. It is crucial that a person using the software BRIDGe can trust it, in other words, BRIDGe should be as reliable as possible, and to obtain this goal, diversified and deep tests should be performed. Under these premises, the Sharon simulator comes very useful it can produce an output that corresponds to the real sensors' activations and so it can be used as input for BRIDGe that will have to analyze it to produce the various statistics that are interesting for the relatives of the dweller. It is important to point out that only the sensors' activations should be used as input for BRIDGe and not the high-level scheduling that BRIDGe can produce which instead should be inferred by BRIDGe, so it is possible to conclude that Sharon can provide also a part of what should be the output of BRIDGe. Another advantage of Sharon is that it uses a model-based approach in the generation of the output which means that the user plays no role in the execution of the algorithm and so its intervention is not needed, moreover has already shown it is possible

to set a parameter in the Main class that regulates how many days of simulation should be generated. This is an interesting advantage because the testing can be completely automated it is possible to create a script that can launch Sharon and at the end of its simulation another script can launch BRIDGe that will take in input the files generated by Sharon containing the sensors' activations. The problem that was presented to us was that the output of Sharon could have made BRIDGe crash.

At the beginning of the simulation, in the Main.java class, there is an if statement that discriminates if the high-level scheduling should be imported or generated. There is a second if statement that discriminates if the sensors' activations should be generated and then another if statement that has the function to choose between the agent generation or the probabilistic generation. It is important to notice that the control flow of the program, independently of the results of the evaluation of the conditions in the if statements, will create one thread that generates or imports the high-level scheduling and if the boolean flag for the generation of the low level is true then also for the low level will be created a thread that will take as a parameter for the constructor function the list of ADLs representing the high-level scheduling. So the objective was to somehow reduce the throughput of output data that could flow from Sharon to BRIDGe. We simply solved this issue by exploiting the ability of a Thread to be set in a state of sleep for a given amount of time, this would have slowed the generation of the output as desired by the user. Going into more details now we will give how we implemented this feature. We created a boolean variable in the Main.java file, in the section where are the other variable that we have already introduced, called SLOWED_GENERATION that if set to true will slow the computation of the low-level thread by putting it into a sleep state, in particular, we have used the method Thread.sleep() of the Thread class. We have created another global integer variable in the Main class that is passed as a parameter to the method function Thread.sleep() and that will represent the number of milliseconds for which the thread can be put in a sleep state so that the user can decide how much the computation should be slowed down.

2.5. Real time data generation

The second problem that was presented to us was that Sharon was not able to produce data in real-time in the sense that it is possible to identify two steps in the computation of the output, there is a high-level step and a low-level step, and each step is computed by a different thread. The high-level thread produced the high-level scheduling composed of ADLs and the low-level thread produced the low-level sensors activations based on

the high-level scheduling. To produce the sensors' activations the list of ADLs is needed but these two elements were computed by two different threads that were created and started in the main function of the Main.java class. According to the implementation of the Thread class in java, it is not possible for the user or the programmer to state which one of the two threads with the same priority is running, but the sequentiality of the execution of the two threads that must be preserved because of the nature of the output. To preserve the sequentiality of the two threads the one which dealt with the computation of the sensors' activations was forced to sleep for a second using the method Thread.sleep(1000) where the parameter "1000" represents the number of milliseconds for which the thread must be in the sleep state.

We partially solved the problem by eliminating the second thread, the one computing the sensors' activations. In particular, we have copied the method run() of the canceled thread and we have created another method in the first thread with that copied code. We have copied in the high-level thread also all the needed variables and the other methods of the low-level thread. We did also some minor, but necessary changes, to have a correct execution. We did this procedure to be able to correctly call the low-level procedure to obtain the low-level dataset at the correct time. We said that we have reached this objective but partially because the feature of calling at the right time the generation of the low-level part of the output is not available when the boolean variable USE_HMM_LL, in the Main class, is set to true, when it is set to true the software generates all the high-level output and then all the low-level output, but when it is set to false for each ADL it can compute the corresponding sensors' activations right after the current ADL has been scheduled.

2.6. Anomalies generation

An important feature that should be implemented to make the simulator more complete and useful is the possibility to generate and represent anomalous, dangerous, unlucky events in the dataset of sensors' activations. Some examples of these types of events can be the fall of the dweller, the fact that the dweller opens the fridge and forgets to close it. The set of possible events that can be represented is limited by the number and the type of the sensors in the virtual environment, these sensors in Sharon can be found in the folder "config/env/sensors", this limit derives from the fact that the abnormal events are represented at the low level in the dataset that contains only the sensors' activations. When the request for a way to generate and represent abnormal events was raised, we were thinking in the wrong direction in the sense that we were imagining all the possible

anomalies that we could have forecast, given the virtual environment, and how we could represent them in the dataset. Later we have found this approach at least limiting if not wrong we were creating a function, for each anomaly, that would output a string representing the sensors' activations appropriately modified to represent the abnormal event. This approach presented some concerning problems: it was not easy to find a lot of abnormal events, it was quite complicated to generalize because each anomaly would have corresponded to a hard-coded algorithm, and it was difficult to estimate the duration time of an anomaly.

We had the idea to give all the power to the user and this was the turning point of the design part of this feature. The main reason that made us think about this approach was the fact that we were not satisfied with the representation of the anomalies in the low-level dataset, they seemed too specific to us and sometimes also a bit naive. We thought that if the simulator would have been used by a person specialized in the study of human behavior, for example, a psychologist, this professional figure would be for sure more capable than us in representing the anomalies in the dataset. As said, we give more power to the user, in the sense that it is the user itself who decides how an anomaly should be represented, when it should be placed in the dataset, and for how long. We wanted to create a way to let the user program the anomaly representation part of the software. Now we will present the changes that we made to the source code of the simulator to obtain this feature. To represent the concept of an abnormal event that lasts over time we have created the class Event.java that contains an array of singular anomalous events that we have called an anomaly, each event object has as a private field a staring time represented by an integer initialized by default to zero and has another private field containing an integer variable that stands for the number of time that the anomalies in the event should be repeated. We have created another class Anomaly.java that as said represents an atomic anomalous event, in other words, it represents a single line of output in the sensors' activation dataset that has been generated according to the specifications provided by the user expert in the field of human behavior. The target user will program the generation and representation part of the simulator by writing in a specific text file a particularly formatted string, this string will contain all the information needed to generate the appropriate anomalous sensors' activations that represent an abnormal event. The user can generate complex input strings and this can be done thanks to the format that we have invented this is a sort of programming feature because the repetition property of an event object acts as a statement in modifying the output dataset and the time field of an event acts like an if statement. Now we will specify the format that should be used to create an input string that is later interpreted by the simulator to produce

part of the output, this will also clarify the previous statement.

Example of a general input string encoding the anomalous events: [(x,y,z)(a,b,c).N;T] Now we will discuss what it means. A couple of [] is used to identify an anomalous event that directly maps to the Event class. It is possible to write a sequence of events by simply writing a sequence of couples of parenthesis. The order of representation of the events does not depend on the order of each event in the sequence. Each couple of () represents an anomaly in the event, in other words, it represents a line in the sensors' activations, particular each couple of () can contain from one to twenty integers, each of which represents the Id of a sensor placed in the virtual environment and because of that it makes no sense to put more than twenty numbers in the comma-separated list and for the same reason, each number should be between zero and nineteen. The effect of putting x,y,z in a couple of round brackets is to have a line in the sensors' activations that has only the sensors which have the same Ids as the ones indicated in the sequence. It is possible to have a couple of round brackets without any character between them, this means that the output line will have all the sensors turned off, that is to say, that all the twenty numbers indicating if a sensor is active or not will be zeros. In a single event it is possible to have more than one anomaly, this means that the two anomalies will appear in the output in series and in this case, the order of appearance in the input string will be replicated in the output dataset. In particular, if the example provided above is considered there will be a line in the output dataset that will have "1" in positions x, y, z, and in all the other positions there will be "0", in the following output line there will be "1" in positions a, b and c and in all the other positions there will be "0". In the string, after the character '.', there is another integer N that specifies how many iterations of the sequence of anomalies should be printed in the output dataset, in particular, the above example specifies that the anomaly (x,y,z) must be printed, then the anomaly (a,b,c) must be printed and this must be done for N consecutive times. The "." character is used just to make the parsing of the string easier during the execution of the software. The character ";" has the same purpose. The integer T represents the time instant at which the anomalous event should be placed in the dataset. This input should be given to the program as a configuration file, in particular, the program expects it in the folder config/Anomalies_list and it should be a text file. There is also a variable in the Main.java class that is named USE_ANOM, it is a boolean that needs to be set to true if the user wants to use the feature that allows the representation of the anomalies in the dataset. We have chosen to force the Sharon simulator to read from a text file the input string containing all the information needed for the representation of the anomalies because this was the pattern already chosen in the development of the code by the guys

that preceded us. To manage the reading of the input from the file we have created an AnomalyDB.java file, the class in this file can read the text from the file and load it into a string, this class has another method that takes as an argument a string that contains the anomalous events coded in the previously specified way and returns as output an array of events where each event has a field the containing an array of the anomalies that make it up, also the other parameters of both the classes Event and Anomaly are set. After having retrieved all the anomalous events from the configuration file, during the high-level part of the computation, if the boolean enabling the anomalies generation is set to true, at each iteration the program checks if there is an event that has a firing time that is equal to the current time, if so the current high-level activity is preserved but there is a function that takes in input the occurring abnormal event and prints in the output file the corresponding anomalies. It is important to point out that the anomalies affect only the low-level dataset which contains sensors' activations, it does not affect the high-level dataset that contains the ADL scheduling, the main reason for this choice is that only the first one should be used by another software to extract information from the data that is producing a dweller in a real smart home environment, moreover the concept of an anomaly is similar to the concept of an unforeseen event and so it would not make sense to represent it in a scheduling dataset. The purpose of this feature is to increase the quality of the datasets that Sharon can produce so that this new information can be used to train machine learning algorithms that are more powerful, in particular, a software like BRIDGe should be able to inform the relatives of the real dweller if something as a fall of the inhabitants has happened, that is to say, that the monitoring software should be able to detect and notify anomalies from the real sensors' activations. To facilitate the development of this feature in the real monitoring software we have decided to double the output, now Sharon, if the user requires it, can generate two datasets containing sensors' activations, one contains the sensors' activations corresponding to the high-level scheduling and in the folder data/SensorOutput it is possible to find a text file for each day of the simulation. We have created another folder in the project to host the new dataset that contains the anomalies and this folder is data/AnomalyOutput and contains the files in the same structure. Each file can contain up to a line for each second of the day, so they are not very human-readable, this can be a problem during the development of a real smart home environment algorithm based on this data, so we have decided to make it a bit more human-readable with respect to the anomaly feature, in fact, each line corresponding to an anomaly in the sensors' activations dataset is composed of two parts, the normal sensors' activations consisting in twenty boolean values and then there is the keyword "anomalous" that denotes that it is a line corresponding to an anomaly.

2.7. Forcing an ADL in the high level scheduling

The next objective that we were given was to be able to force, during the execution of the simulation, the scheduling of a specific ADL at a certain instant and it should have been possible for the user to set both these parameters at run time. The change in the schedule should be reflected also in the dataset containing the sensors' activations.

There are fourteen different activities that can be be scheduled, and they are:

- Sleeping which has an ID equal to 1.
- Breakfast which has an ID equal to 2.
- Working which has an ID equal to 3.
- Lunch which has an ID equal to 4.
- Dinner which has an ID equal to 5.
- Shower which has an ID equal to 6.
- Toilet which has an ID equal to 7.
- WatchingTV which has an ID equal to 8.
- Cleaning which has an ID equal to 9.
- GoingOut which has an ID equal to 10.
- Internet which has an ID equal to 11.
- Reading which has an ID equal to 12.
- Relax which has an ID equal to 13.
- Other which has an ID equal to 14. To be able to use this feature it is crucial to know the ID of each ADL, it is possible to find all the information regarding each ADL in the folder config of the project Sharon where there is a configuration file for each activity that starts with the name that is reported in the list above. The list provided can be used as a quick reference to check the ID of a particular ADL.

To implement this feature we exploited the fact that the simulation is launched via terminal in which during the simulation are shown various data. We decided to implement a simple but effective dialog that asks the user for the times in which he or she wants to stop the high-level execution of the program to manually choose the ADL to perform in that specific time. Though there is a small problem with this solution when the user

types in the terminal the characters that are typed are not shown and this can cause the user to make some mistakes and so cause a wrong execution of the program. To partially limit this weakness, we decided to print the input that the user has just typed so that he or she can check if that input is correct. In particular, after having launched the program, before starting the simulation, the user will be asked to enter a comma-separated list of time instants in which the simulation should be stopped and in that times, again with a dialogue in the terminal, will be asked to enter the ID of the ADL that he or she wants to force. The program will stop for every time specified in the comma-separated list. Speaking about the source code, when the simulation of the high level is iterating over all the seconds of the simulation days, if the second matches one of the times that was given by the user in the comma-separated list, then the execution stops and asks the user to submit an integer number that corresponds to an ADL ID, after the input is submitted the program retrieves the corresponding ADL from its ID, then it is set as the ongoing ADL and its elapsed time is set to zero, subsequently the needs are updated taking into account the current ADL, and if the user wants there is a boolean that can enable or disable the fact that this forced ADL can affect or not the output of the sensors' activations, if the boolean is set to false then the forced ADL will only affect the high-level output containing the schedule of the ADLs. The user can choose to not use this feature by simply pressing "enter" when asked to insert the comma-separated list of times. To facilitate the development of new applications through the use of the datasets produced by Sharon we have decided that the lines in the high-level output regarding the forced ADLs end with a default keyword that in this case is "forced activity".

At this point, after having implemented a feature that would involve the interaction with the user at run time, we were asked to extend this also to the anomaly generation and we did it. In particular at the beginning of the simulation, just after the dialogue in the terminal where the user is asked to insert a comma-separated list of times, another dialogue requires the user to insert a string with the same format of the string present in the text file Anomalies_list, that is the format that has been discussed in the part dedicated to the anomalies generation. It is possible to disable this feature by simply pressing the enter key when asked to insert the string.

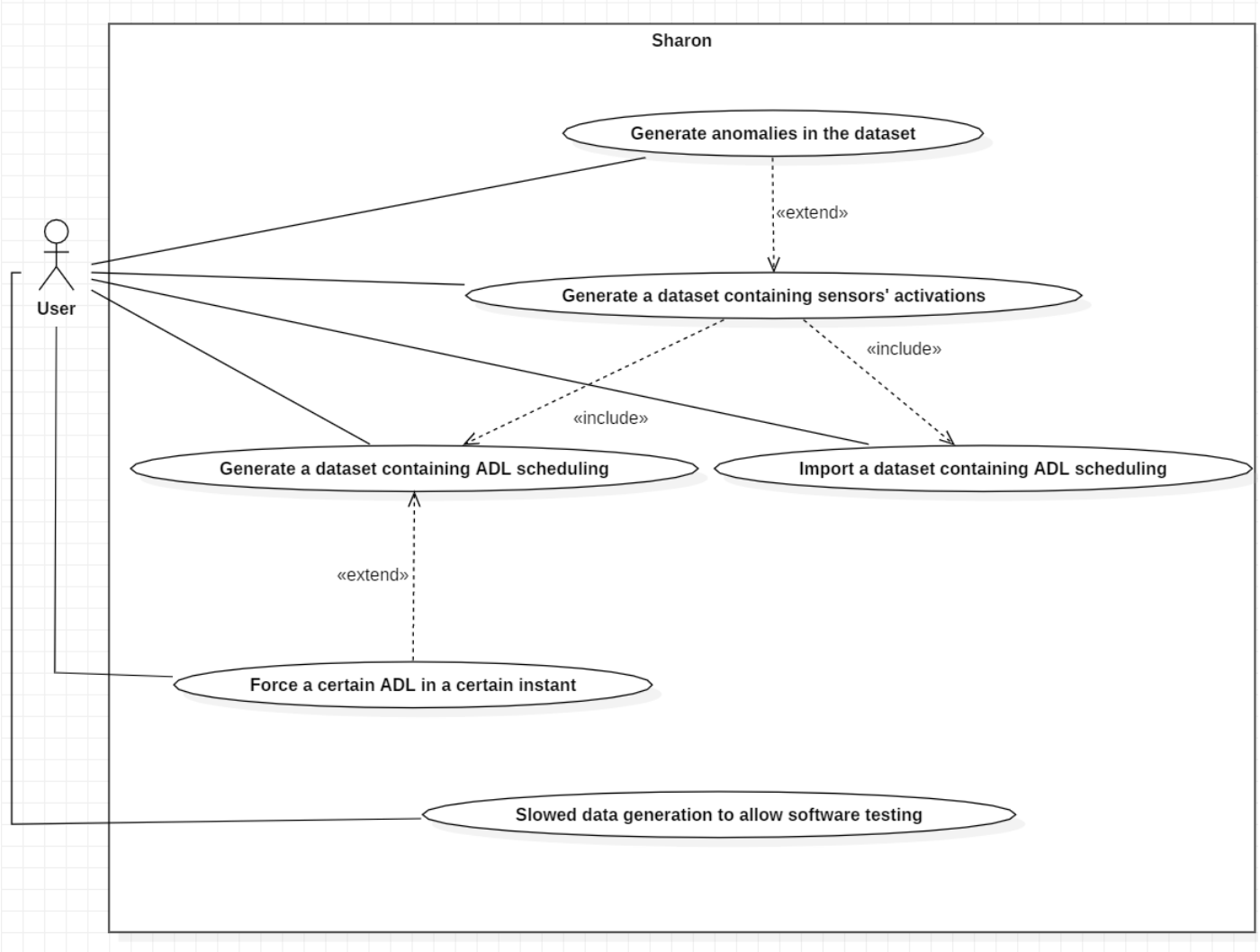

Figure 2.1: A use case diagram that summarises the functionalities of Sharon.

3 | Results

In this section, we will illustrate the result of our work. In particular, this has been an implementation work that has added some key features to a project that has been developed at Politecnico di Milano. Because of this nature of the work, also the results will be presented appropriately, that is we will demonstrate that with the changes that we have done, it is possible to achieve our goals. As far as possible we will provide details on how to replicate the results that will be shown. However, it is not possible to replicate exactly the shown results as the simulator is based on a model that is also based on pseudo-random calculations. Despite this inconvenience, the experiments, from which the results have been taken, have been thought to mitigate this issue.

To gain some results there must be an experiment. In computer science, the software can be thought of as an experiment, so we wrote a computer program, external to Sharon, that can take in input the output of the simulator, process it, and then return some statistics that are tailored to highlight the outcome of the implementation process.

In particular, this program, written in Java, has four subfolders in its root directory, each of which contains a specific dataset that has been generated through Sharon but with different parameters. One directory contains the high-level scheduling of the ADLs, one holds the high-level scheduling of the ADLs but with some forced ADLs according to a specific input that will be specified later, one encompasses the low-level sensors' activations, one includes the low-level sensors' activations that have been created using a specific input, to represent anomalous behavior, that will be made clear after. The program reads from these folders and computes only the statistics that are relevant to demonstrate the success of the implementation, which has been described in this paper. If the reader wants to see how Sharon performs in the generation of the high-level scheduling and the low level, we refer to reading the work by Veronese, Masciadri, Trofimova, Matteucci, Salice [25], which in the result section, shows various metrics regarding probability distances between distributions. The program used to inspect the output of Sharon will not be available.

3.1. Results regarding the high level scheduling

Now we will introduce the resulting statistics that the program, that tests Sharon, can provide to the user. We start from the statistics computed from the high-level dataset. To summarize, the main change that has been made to the high-level part of Sharon regards the possibility to force a particular ADL in a certain instant. We thought to compute the number of times that a certain ADL is done by the virtual agent and the relative frequency of that ADL with respect to all the ADLs performed. After having written the code to do this operation, we used it to inspect the data representing the ADLs not forced by the user. We noticed that the ADL "Working", which has an ID equal to 3, was never executed.

```
COUNTS
The ADL_ID is: 1 and its counter is: 192
The ADL_ID is: 2 and its counter is: 80
The ADL_ID is: 3 and its counter is: 0
The ADL_ID is: 4 and its counter is: 145
The ADL_ID is: 5 and its counter is: 124
The ADL_ID is: 6 and its counter is: 91
The ADL_ID is: 7 and its counter is: 316
The ADL_ID is: 8 and its counter is: 174
The ADL_ID is: 9 and its counter is: 48
The ADL_ID is: 10 and its counter is: 225
The ADL_ID is: 11 and its counter is: 104
The ADL_ID is: 12 and its counter is: 91
The ADL_ID is: 13 and its counter is: 171
The ADL_ID is: 14 and its counter is: 7
1768


FREQUENCIES
The ADL_ID is: 1 and its frequence is: 0.1085972850678733
The ADL_ID is: 2 and its frequence is: 0.04524886877828054
The ADL_ID is: 3 and its frequence is: 0.0
The ADL_ID is: 4 and its frequence is: 0.08201357466063348
The ADL_ID is: 5 and its frequence is: 0.07013574660633484
The ADL_ID is: 6 and its frequence is: 0.051470588235294115
The ADL_ID is: 7 and its frequence is: 0.17873303167420815
The ADL_ID is: 8 and its frequence is: 0.09841628959276018
The ADL_ID is: 9 and its frequence is: 0.027149321266968326
The ADL_ID is: 10 and its frequence is: 0.12726244343891402
The ADL_ID is: 11 and its frequence is: 0.058823529411764705
The ADL_ID is: 12 and its frequence is: 0.051470588235294115
The ADL_ID is: 13 and its frequence is: 0.09671945701357466
The ADL_ID is: 14 and its frequence is: 0.003959276018099547
```

Figure 3.1: A screenshot representing the high level statistics.

This fact can be seen in the first image. Leaving aside considerations on why this happened, we thought to exploit this fact to show the goodness of our work. In particular, we thought that to prove that the user can force some ADLs, we could make a specific input to force only the ADL that was not appearing in the normal dataset, this would have made the changes in the statistics a clear proof of the good result of our implementation. To obtain that result, the input that the user should give to Sharon at run time, after having set the boolean representing the will of using this specific feature, is the following string, when asked about the time instants 1, 90000, 180000, 270000, 360000, 450000, 600000, 690000, 800000, 890000, 900000, 1000000, 11000000, 12000000, 2050000, 21000000, 2190000, 2200000, 2250000, 45000, 600000, 800000, 1500000, 3000000 (without spaces, here present for readability) where each number represents the time instant of the simulation at which the program will ask the user to insert the ID of the ADL, that needs to be forced. The time instants were picked up randomly. To replicate this particular experiment, the user has to insert the number 3 every time that is asked to provide the ID of the ADL to be forced. If the reader wants to replicate the results shown in the second image, the previous steps must be followed, in particular, only the counter and the frequencies for the ADL having ID equals to 3 will be perfectly replicated because decided by the user, instead of the statistics regarding the other ADLs will not be the same, because of the random elements of the model.

```
COUNTS
The ADL_ID is: 1 and its counter is: 200
The ADL_ID is: 2 and its counter is: 82
The ADL_ID is: 3 and its counter is: 21
The ADL_ID is: 4 and its counter is: 148
The ADL_ID is: 5 and its counter is: 130
The ADL_ID is: 6 and its counter is: 87
The ADL_ID is: 7 and its counter is: 336
The ADL_ID is: 8 and its counter is: 164
The ADL_ID is: 9 and its counter is: 47
The ADL_ID is: 10 and its counter is: 246
The ADL_ID is: 11 and its counter is: 127
The ADL_ID is: 12 and its counter is: 95
The ADL_ID is: 13 and its counter is: 168
The ADL_ID is: 14 and its counter is: 7
1858


FREQUENCIES
The ADL_ID is: 1 and its frequence is: 0.10764262648008611
The ADL_ID is: 2 and its frequence is: 0.04413347685683531
The ADL_ID is: 3 and its frequence is: 0.011302475780409042
The ADL_ID is: 4 and its frequence is: 0.07965554359526372
The ADL_ID is: 5 and its frequence is: 0.06996770721205597
The ADL_ID is: 6 and its frequence is: 0.04682454251883746
The ADL_ID is: 7 and its frequence is: 0.18083961248654468
The ADL_ID is: 8 and its frequence is: 0.08826695371367062
The ADL_ID is: 9 and its frequence is: 0.02529601722282024
The ADL_ID is: 10 and its frequence is: 0.13240043057050593
The ADL_ID is: 11 and its frequence is: 0.06835306781485469
The ADL_ID is: 12 and its frequence is: 0.051130247578040904
The ADL_ID is: 13 and its frequence is: 0.09041980624327234
The ADL_ID is: 14 and its frequence is: 0.003767491926803014
```

Figure 3.2: A screenshot representing the statistics after that the dataset has been modified forcing some ADLs.

It should be noticed that if in the string representing the list of timestamps, a number is repeated more than one time, then this duplicate will be present also in the generated dataset. Another observation that is worth doing is that if the typed number is greater than the product between the number of simulated days and the number of seconds per day, then the user will not be asked to provide an ID for a certain ADL to be forced in that particular timestamp. The third image shows a subset of the dataset representing the high-level scheduling. In particular, it is possible to notice that there are only ADLs with the same ID, equal to 3, representing the activity of "working".

```
Here are the forced lines
1,3,working,0:0 forced activity
45000,3,working,12:30 forced activity
90000,3,working,1:0 forced activity
180000,3,working,2:0 forced activity
270000,3,working,3:0 forced activity
360000,3,working,4:0 forced activity
450000,3,working,5:0 forced activity
600000,3,working,22:40 forced activity
600000,3,working,22:40 forced activity
690000,3,working,23:40 forced activity
800000,3,working,6:13 forced activity
800000,3,working,6:13 forced activity
890000,3,working,7:13 forced activity
900000,3,working,10:0 forced activity
1000000,3,working,13:46 forced activity
1500000,3,working,8:40 forced activity
2050000,3,working,17:26 forced activity
2190000,3,working,8:20 forced activity
2200000,3,working,11:6 forced activity
2250000,3,working,1:0 forced activity
3000000,3,working,17:20 forced activity
```

Figure 3.3: This screenshot shows a subset of the dataset containing only the forced ADLs, it is important to notice the timestamps.

This image not only is further proof of the fact that the forced ADL is the one that the user wanted, but it also shows that also the time stamps are correct, which means that the ADLs are where the user wants them to be in the final dataset.

3.2. Results regarding the low level scheduling

Now we will describe the results that we have found about the low-level dataset, containing sensors' activations. It is important to recall some ideas. The low-level dataset is divided into text files, each of which represents a simulated day, and each row of each day represents the sensors' activations at a certain second of the day. The most relevant feature that we implemented to enrich the power of the simulator is the fact that the user can specify a string in a specific input file, that contains formatted instruction on how to produce a subset of the sensors' activations. It is possible to insert the string also at run time, without modifying the corresponding text file. Given the same input, the two methods will produce the same output. This feature should be used by the simulator's user to reproduce anomalous behavior in the virtual dweller.

We thought to start analyzing with the auxiliary tool the lower level dataset containing

sensors' activations without anomalies. This dataset is composed of twenty zeros or ones, representing active or sleeping sensors, and two more numbers that are not relevant in the discussion. The most rational approach to adopt was to reproduce the previous strategy used in the high-level case, but this time the object of the experiment is the sensors and not the ADLs. Hence we decided to scan the dataset to build one statistic representing the total number of times that a certain sensor is activated and another statistic representing the relative frequency, which is the total number of times in which that sensor is active over the total amount of timestamps of the simulation. After having generated a dataset without anomalies with Sharon, we analyzed it with the tool and we found the results that are shown in picture four. In this picture, the reader can see that in the first part there are, for each sensor represented by its ID, the various counters, in the second part of the image there are the frequencies displayed with the same style.

```
Here are the sensors' counters
Sensor ID: 1 , counter: 10576
Sensor ID: 2 , counter: 0
Sensor ID: 3 , counter: 51767
Sensor ID: 4 , counter: 1316967
Sensor ID: 5 , counter: 13819
Sensor ID: 6 , counter: 187409
Sensor ID: 7 , counter: 446200
Sensor ID: 8 , counter: 57426
Sensor ID: 9 , counter: 41543
Sensor ID: 10 , counter: 0
Sensor ID: 11 , counter: 0
Sensor ID: 12 , counter: 2177576
Sensor ID: 13 , counter: 284752
Sensor ID: 14 , counter: 16930
Sensor ID: 15 , counter: 2301592
Sensor ID: 16 , counter: 106037
Sensor ID: 17 , counter: 28685
Sensor ID: 18 , counter: 151596
Sensor ID: 19 , counter: 132513
Sensor ID: 20 , counter: 2982340
Here are the frequencies of each sensor
The frequency for the sensor 1 is 0.0010260262979387893
The frequency for the sensor 2 is 0.0
The frequency for the sensor 3 is 0.005022154251644979
The frequency for the sensor 4 is 0.12776501281368696
The frequency for the sensor 5 is 0.0013406446114992557
The frequency for the sensor 6 is 0.018181407192739273
The frequency for the sensor 7 is 0.04328790980902872
The frequency for the sensor 8 is 0.005571159813297363
The frequency for the sensor 9 is 0.004030277089189781
The frequency for the sensor 10 is 0.0
The frequency for the sensor 11 is 0.0
The frequency for the sensor 12 is 0.2112566416187932
The frequency for the sensor 13 is 0.027625098372793696
The frequency for the sensor 14 is 0.0016424569992533756
The frequency for the sensor 15 is 0.22328800294303458
The frequency for the sensor 16 is 0.010287136020663331
The frequency for the sensor 17 is 0.002782863498144305
The frequency for the sensor 18 is 0.01470702370105226
The frequency for the sensor 19 is 0.012855694290730217
The frequency for the sensor 20 is 0.2893304906765099
```

Figure 3.4: This screenshot shows the statistics for the sensors' activations without anomalies.

The reader can see that there are some counters equal to zero, in particular, the sensors 2, 10, and 11 are never activated during this simulation. With the same reasoning as before, we thought that even in this case we could exploit this fact to present proof of the goodness of our work. Hence we built a specific text file representing anomalies that should trigger just those sensors. In particular the string that should be written in the configuration file is the following [(2,10).4;30000][(2,10).7;95000][(2,10).9;1500000][(2,10).3;2000000] [(2,10).5;3000000][[(2,10).9;450000][[(2,10).5;520000] [[(2,10).5;610000][[(2,10).5;700000]. The reader can see that we have chosen to trigger only the sensors 2 and 10, and not the num-

ber 11 because we also want to see if the situation in which all those three sensors are zero will happen again. After having produced a new low-level dataset with sensors' activations and anomalies, we analyzed it using the other program. It is possible to see the new results in the following image.

```
Here are the sensors' counters
Sensor ID: 1 , counter: 26286
Sensor ID: 2 , counter: 52
Sensor ID: 3 , counter: 103438
Sensor ID: 4 , counter: 2594800
Sensor ID: 5 , counter: 42883
Sensor ID: 6 , counter: 362061
Sensor ID: 7 , counter: 897042
Sensor ID: 8 , counter: 120536
Sensor ID: 9 , counter: 84780
Sensor ID: 10 , counter: 52
Sensor ID: 11 , counter: 0
Sensor ID: 12 , counter: 4370248
Sensor ID: 13 , counter: 572835
Sensor ID: 14 , counter: 33456
Sensor ID: 15 , counter: 4628512
Sensor ID: 16 , counter: 221557
Sensor ID: 17 , counter: 58451
Sensor ID: 18 , counter: 306438
Sensor ID: 19 , counter: 276756
Sensor ID: 20 , counter: 5949611
Here are the frequencies of each sensor
The frequency for the sensor 1 is 0.0012729424806852796
The frequency for the sensor 2 is 2.5181849271716706E-6
The frequency for the sensor 3 is 0.005009154086476601
The frequency for the sensor 4 is 0.12565742786586637
The frequency for the sensor 5 is 0.002076679312151976
The frequency for the sensor 6 is 0.01753339524839812
The frequency for the sensor 7 is 0.04344072391230634
The frequency for the sensor 8 is 0.005837152661183932
The frequency for the sensor 9 is 0.00410560996395412
The frequency for the sensor 10 is 2.5181849271716706E-6
The frequency for the sensor 11 is 0.0
The frequency for the sensor 12 is 0.21163639695388728
The frequency for the sensor 13 is 0.027740470437622768
The frequency for the sensor 14 is 0.001620161440835681
The frequency for the sensor 15 is 0.22414325295448467
The frequency for the sensor 16 is 0.010729259575180266
The frequency for the sensor 17 is 0.0028305851380406024
The frequency for the sensor 18 is 0.014839760629089085
The frequency for the sensor 19 is 0.013402361302006208
The frequency for the sensor 20 is 0.28811962966797633
```

Figure 3.5: A screenshot that shows the low level statistics of a dataset with anomalies.

As we can see the counters for sensors 2 and 10 are changed, but for sensor 11 is still zero. If the reader counts all the numbers in the string that is between the character '.' and ';', he/she will notice that their sum is 52. It is also possible to see that the counter for sensor 12 has remained the same and it is again equal to zero. This can be considered

as proof of the implementation process. It is necessary to verify that the anomalies are placed at the right moment. For this scope, we invite the reader to watch the following image in which there is a list containing all the sensors' activations that correspond to an anomaly.

```
Here are the lines with an anomaly and the timestamp
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 8 1 anomalous 30000
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 8 1 anomalous 30001
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 8 1 anomalous 30002
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 8 1 anomalous 30003
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 95004
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 95005
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 95006
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 95007
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 95008
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 95009
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 95010
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 450011
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 450012
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 450013
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 450014
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 450015
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 450016
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 450017
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 450018
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 450019
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 520020
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 520021
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 520022
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 520023
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 520024
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 610025
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 610026
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 610027
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 610028
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 610029
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 700030
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 700031
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 700032
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 700033
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 700034
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 10 1 anomalous 1500035
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 10 1 anomalous 1500036
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 10 1 anomalous 1500037
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 10 1 anomalous 1500038
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 10 1 anomalous 1500039
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 10 1 anomalous 1500040
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 10 1 anomalous 1500041
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 10 1 anomalous 1500042
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 10 1 anomalous 1500043
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 2000044
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 2000045
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 anomalous 2000046
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 11 1 anomalous 3000047
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 11 1 anomalous 3000048
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 11 1 anomalous 3000049
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 11 1 anomalous 3000050
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 11 1 anomalous 3000051
```

Figure 3.6: A screenshot that shows a subset of the low level dataset plus the timestamp of each line.

The reader can check that the timestamps are correct by looking at the number after the character ';' in the input string and the timestamps in the image, this is further proof of the good outcome of the implementation phase.

4 | Conclusions and future works

In this section, we will present the conclusions deriving from the results of the previous section. The results, on which this chapter is built, have been derived from experiments that were conceived to inspect the outcome of the implementation work that is the subject of this work. It is straightforward to state that the results are relevant because they are derived from ad-hoc experiments and so it will be possible to determine if the implementation phase has been completed correctly. To state if the results are original or not it is necessary to remember the literature review that has been done in the state of the art chapter. In this case, results can be original if the experiment that has made them is original, which means that the implemented feature, the object of the inquiry of the experiment, must be original. A feature of the software will be considered original if not already implemented in any of the analyzed papers used to write the state of the art section. This is the reason why it is necessary to remember the features of the various simulators presented in the state of the art, or at least the literature review.

Now we focus on the results, that the ad-hoc testing program that we have created, has provided us. It is necessary to notice that in the testing section only one instance of the testing experiment is reported to avoid redundancy, but actually, the actual smart home environment simulator has been tested with several random inputs for the same feature.

For the high-level part, we can assert that we have reached the goal that we had set for ourselves, which consisted in allowing the user to decide at run time which ADL force in a certain time of the simulation. According to the results shown in the appropriate section, we can assert that the objective has been completely reached because the differences in the output statistics between the dataset with the forced ADLs and the normal dataset completely respect our previsions. The low-level feature consisted in designing a particular format for encoding a set of anomalous events to be reproduced in the low-level dataset containing sensors' activations. Also in this case the difference between the dataset with anomalies and the dataset without them are in line with our expectations, so even in this case, we can declare that we reached our goal. The more abstract goal of modifying Sharon so that it would allow the user to do software testing of a real smart home environment

application has been partially reached. In particular, the feature that allows the user to set the pace of the simulation works as expected, but the fact that Sharon can produce the two datasets in parallel works only if the low level operates under the "agent assumption" which means that a virtual agent is simulated moving in the space triggering the sensors, instead of the low level operates under the "probabilistic assumption" which means that given an ADL the sensors are triggered using probability distributions, then, in this case, this feature does not work and the generation of the two datasets is sequential.

The feature that allows the execution of a certain ADL in a certain time can be thought to be original because there is no simulator that at the run time asks the user to provide the activity to be executed by the virtual dweller. In particular, in this case, it is like we are mixing the model-based approach with the interactive-based approach, the first is represented by how Sharon works, and the second is featured by the fact that the user directly interacts with the simulation by specifying the ADL. We can state that this is an original result and will allow the user of the simulator to represent anomalous sequences of actions. The other feature allows the user to specify a sequence of events composed of anomalies that are reproduced in the sensors' activations dataset. This result is original in the sense that no read paper ever talks about the representation of anomalous events, so we can assert that we have implemented a breakthrough functionality.

During the testing phase, when we were executing the experiments, we discovered a strange phenomenon in the output dataset, both in the high-level dataset containing ADLs and in the low-level dataset containing sensors' activations. In particular, looking at the statistics of the high-level dataset we can notice that the ADL with ID equal to 3, which represents the activity of working, is never executed by the virtual agent. This is not realistic because we have simulated 90 days and it is not likely that a person does not work a single time in such a time interval. Also in the low-level dataset, if we look at the sensors' activations statistics, we can notice that three sensors never get triggered, these sensors have IDs 2,10,11. In particular, we can notice that the sensor with ID 10 is placed on the wardrobe and the sensor with ID 11 is placed on the bathroom cabinet. Even in this case, we can say that it is not realistic that those sensors are never triggered if we consider the time interval of our simulation. The scope of this thesis was not to assert or improve the quality of the output datasets, but we think that it is important to signal this small problem.

We will now talk about future works that we think could improve the Sharon simulator. We think that in light of the results that have shown a strange behavior regarding the fact that the ADL with ID equal to 3 is never executed and the sensors with IDs equal to 2,10 and 11 are never triggered. So the first work that we would suggest is the one that

checks the correctness of the models producing the datasets. The second suggestion that we would like to give arises from the analysis of the state of the art. In particular, we have highlighted the fact that various simulators make available to the users various tools that allow the creation of an ad-hoc smart-home environment in which the simulation takes place. We think that Sharon should provide the user with a simple tool to create a virtual environment in which to place the various sensors. A less important feature that could be added is a graphical interface that could allow the user to see the actions of the agent in a friendly way during the simulation itself; we think that this would be particularly useful if Sharon is used to doing online software testing because the user could see in real-time and in an immediate way the correlation between the simulated data and the reported data.

Bibliography

[1] *Activity and Assistive Program*. URL http://www.aal-europe.eu/wp-content/uploads/2015/11/20151001-AAL-Strategy_Final.pdf.

[2] Ageing and healt. *World Health Organization website*. URL https://www.who.int/news-room/fact-sheets/detail/ageing-and-health.

[3] World population ageing. *Department of Economic and Social Affairs, Population Division*, 2017. URL https://ifa.ngo/wp-content/uploads/2018/12/WPA2017_Highlights.pdf.

[4] World population ageing 2019: Highlights. *Department of Economic and Social Affairs, Population Division*, 2019. URL https://www.un.org/en/development/desa/population/publications/pdf/ageing/WorldPopulationAgeing2019-Highlights.pdf.

[5] Population ages 65 and above (% of total population). *World Bank Indicators*, 2019. URL https://data.worldbank.org/indicator/SP.POP.65UP.TO.ZS.

[6] A. Ariani, S. J. Redmond, D. Chang, and N. H. Lovell. Simulation of a smart home environment. pages 27–32, 2013. doi: 10.1109/ICICI-BME.2013.6698459.

[7] I. Armac and D. Retkowitz. Simulation of smart environments. pages 322–331, 2007. doi: 10.1109/PERSER.2007.4283934.

[8] J. Bruneau, W. Jouve, and C. Consel. Diasim: A parameterized simulator for pervasive computing applications. pages 1–10, 2009. doi: 10.4108/ICST.MOBIQUITOUS2009.6851.

[9] P. Eunil, C. Yongwoo, H. Jinyoung, and J. K. Sang. Comprehensive approaches to user acceptance of internet of things in a smart home environment. *IEEE Internet of Things Journal*, 2017. URL https://ieeexplore.ieee.org/abstract/document/8031000.

[10] C. Fabien, B. Simon, B. David, and M. S. Hawley. Simulation of home daily activities for lifestyle monitoring systems development. *IEEE Trans-*

action on Information Technology in Biomedicine, 2010. URL https://www.researchgate.net/publication/45196532_Simulation_of_home_daily_activities_for_lifestyle_monitoring_systems_development.

[11] S. Helal, J. W. Lee, S. Hossain, E. Kim, H. Hagras, and D. Cook. Persim - simulator for human activities in pervasive spaces. pages 192–199, 2011. doi: 10.1109/IE.2011.34.

[12] S. Jonathan, N. Chris, and J. Paul. Simulation of smart home activity datasets. *National library of medicine*, 2015. URL https://www.researchgate.net/publication/278790342_Simulation_of_Smart_Home_Activity_Datasets.

[13] B. Kormányos and B. Pataki. Multilevel simulation of daily activities: Why and how? pages 1–6, 2013. doi: 10.1109/CIVEMSA.2013.6617386.

[14] B. Kévin, A. Amir, B. Bruno, and B. Abdenour. Simact: a 3d open source smart home simulator for activity recognition with open database and visual editor. *International Journal of Hybrid Information Technology (IJHIT)*, 2012. URL https://www.researchgate.net/publication/230646951_SIMACT_a_3D_Open_Source_Smart_Home_Simulator_for_Activity_Recognition_with_Open_Database_and_Visual_Editor.

[15] J. Lundström, J. Synnott, E. Järpe, and C. D. Nugent. Smart home simulation using avatar control and probabilistic sampling. pages 336–341, 2015. doi: 10.1109/PERCOMW.2015.7134059.

[16] J. Marcin Pawel, E. Thomas, G. Alexandros, F. Kensuke, E. Sofia, H. Dagmar, G. Julie, K. Sara, S. Osamu, T. Kazuhiko, T. Töres, D. Nannan, K. Fumiko, W. Ryugo, S. Giles Bruno, Y. Makoto, and P. Jian. Ageing and population shrinking: implications for sustainability in the urban century. *npj Urban Sustain*, 2021. URL https://www.nature.com/articles/s42949-021-00023-z.

[17] B. Mario, K. Werner, and K. Josef. A simulator for generating and visualizing sensor data for ambient intelligence environments. 2011. URL https://www.sciencedirect.com/science/article/pii/S1877050911003395.

[18] A. Masciadri, F. Veronese, S. Comai, I. Carlini, and F. Salice. Disseminating synthetic smart home data for advanced applications. 01 2018.

[19] A. Mendez-Vazquez, S. Helal, and D. J. Cook. Simulating events to generate synthetic data for pervasive spaces. *Semantic scholar*, 2008.

[20] N. Noury and T. Hadidi. Computer simulation of the activity of the elderly person

living independently in a health smart home. *Computer methods and programs in biomedicine*, 108 3:1216–28, 2012.

[21] J. Park, M. Moon, S. Hwang, and K. Yeom. Cass: A context-aware simulation system for smart home. pages 461–467, 2007. doi: 10.1109/SERA.2007.60.

[22] M. P. Poland, C. D. Nugent, W. Hui, and C. Liming. Development of a smart home simulator for use as a heuristic tool for management of sensor distribution. *National library of medicine*, 2009.

[23] L. Ronals and M. Andrew. Cost of aging. *Finance and Development*, 2017. URL `https://www.imf.org/external/pubs/ft/fandd/2017/03/lee.htm`.

[24] F. Veronese, S. Mangano, S. Comai, M. Matteucci, and F. Salice. Indoor activity monitoring for mutual reassurance. 2017. URL `https://re.public.polimi.it/handle/11311/1031854?mode=full.593#.YZZZBGDMJPZ`.

[25] F. Veronese, A. Masciadri, A. Trofimova, M. Matteucci, and F. Salice. Realistic human behaviour simulation for quantitative ambient intelligence studies. *Technology and Disability*, 28:159–177, 01 2017. doi: 10.3233/TAD-160453.

List of Figures

List of Symbols

Variable	Description	SI unit
$\boldsymbol{\sigma}_n$	need status	pure number
$\boldsymbol{\gamma}_n$	growth rate	pure number
$\boldsymbol{\zeta}_n$	effect on the need	pure number
$\boldsymbol{w}_{n,a}$	weight of the model	pure number
$\boldsymbol{\Theta}(\boldsymbol{t})_a$	likelihood function	pure number
$\boldsymbol{\nu}$	random variable	pure number
$\boldsymbol{S}(\boldsymbol{a},\boldsymbol{t})$	scoring function	pure number

Acknowledgements

Grazie a tutti.