Real-Time Anomaly Detection in Elderly Behavior with the Support of Task Models

AUTHOR 1, x, x
AUTHOR 2, x, x
AUTHOR 3, x, x

With today's technology, elderly can be supported in living independently in their own homes for a prolonged period of time. Monitoring and analyzing their behavior in order to find possible unusual situation helps to provide the elderly with health warnings and convince them towards positive changes in their behavior with persuasive suggestions at the proper time. Current studies are focusing on the elderly daily activity and the detection of anomalous behaviors aiming to provide the older people with remote support. To this aim, we propose a real-time solution which models the user daily behavior using a task model specification, and a context manager to detect relevant contextual events occurred in their life. In addition, by a systematic validation through a system that automatically generates wrong sequences of events, we show that our algorithm is able to find behavioral deviations from the expected behavior at different times by considering the extended classification of the possible deviations. Moreover, it should enable suitable actions addressing the identified anomaly which can contribute to improving the elderly well-being.

CCS Concepts: • **Human-centered computing** → *HCI theory, concepts and models*;

Additional Key Words and Phrases: Elderly behavior analysis, deviations in task performance, ambient assisted living

ACM Reference Format:
Author 1, Author 2, and Author 3. YYYY. Real-Time Anomaly Detection in Elderly Behavior with the Support of Task Models . *ACM Trans. Web* 0, 0, Article 0 (YYYY), 18 pages. https://doi.org/0000000.0000000

1 INTRODUCTION

The increase of the average age of the European citizens is a current matter of concern as in the next future there will be the need of providing adequate and sustainable support to an increasing number of people who are more subject to suffer from health diseases. Today, remote care applications are available to support remote assistance and help elderly by continuously providing relevant information to the relatives, the caregivers and also elderly themselves. In many cases, older people prefer to have an independent life in their own homes [29], however, support is needed to continue their everyday living routines. Current technologies offer plenty of interactive devices, smart objects, and sensors to detect a wide range of environmental and user-related parameters. However, it is challenging to extract relevant knowledge in real-time from information provided by sensors and devices. In Ambient Assisted Living (AAL) scenarios, monitoring such parameters will be useful to build knowledge about the context around the elderly and detect anomalies and significant changes in their behavior, with the assumption that these anomalies may signal health related problems. Moreover, increasing the elderly autonomy and assisting them in carrying out

Authors' addresses: Author 1, x, x, x, x, x, x, x; Author 2, x, x, x, x; Author 3, x, x, x, x, x, x, x.

1559-1131/YYYY/0-ART0 $15.00
https://doi.org/0000000.0000000

their activities of daily living (i.e. activities that constitute daily routines, such as walking, eating, bathing, dressing, cooking, etc.) create the possibility to extend the time older people can live in their home environment. Nowadays there are several contributions focusing on the elderly because of the numerous health care needs, and to address the behavior deviations and their consequences. However, solutions for detecting “unusual” behavior in the user daily living activities based on the user's plan are missing [9, 17], as well as tools for analyzing such data and acting consequently when certain conditions are met. Meanwhile, mostly these solutions need training data or are narrowed by an offline method [18] for the anomaly detection. It is fundamental that data analysis, both for short-term (alerts generation) and long-term (behavioral analysis) purposes, can be performed taking into account the specificities of the context where the system is used in order to fit the particular needs, requirements and routines/tasks of the considered user. In the same way, the approach to behavior analysis and finding the unusual behaviors should consider that different people may have different habits (e.g., the time they get up or go to sleep, the number and time of meals between various users, their special medical care they need, etc.). In addition, the user monitoring can be beneficial at various levels: in the short-time it allows raising alarms as soon as the elderly behaves unexpectedly, such as when they open the entrance door in the middle of the night; the long-time monitoring is instead aimed to make assessments on relevant deviations from the expected pattern, such as changes in level of physical activity. To address the challenge of keeping older people healthy, fulfilled and independent in their home for as long as possible, we propose a solution to identify anomalies using task model specifications for modeling the “normal” behavior and a Context Manager to detect relevant contextual events occurred in daily life. Later, based on the anomaly type, the method chooses the best way to persuade the elderly to change their anomalous behavior. Consequently, it is possible to promptly act upon any specific detection in a personalized manner in order to provide relevant information. Analyzing and understanding the unusual behavior in the user behavior, not only helps elderly to receive all the relevant warnings but is also highly informative for the caregivers and can be even life-saving in particular situations. On the other hand, applying persuasive techniques increases the chance of acceptance of the proposed interventions by older adults.

We build our work on previous studies [18] but we propose a new algorithm able to address a wider set of cases. While, previous work only considered anomalies offline at the end of each day (i.e., when a complete event sequence is available) and a limited set of anomalies (namely “*Less*”, “*More*”, “*Difference*”). In this paper, we propose a novel algorithm that detects anomalies at real-time (i.e., it operates also on partial, incomplete sequences), and provides a more detailed information about the anomalies detected. The novel algorithm can be exploited into an existing framework for elderly monitoring, which in particular, comprises a Context Manager and a CTT Task model simulator. In order to connect the new algorithm to the framework, we also considered task-related aspects/attributes (e.g., task time and duration, task criticality level, etc.). Each time a new observation is produced by a sensor, the Context Manager identifies one (or even more) corresponding events; in turn, these events immediately feed the Anomaly Detection Algorithm. Due to its ability to operate in real-time, the new algorithm ensures that the anomaly is identified as soon as the event which triggers a deviation, is detected. This document is structured as follows: after the Introduction (Section 1), Section 2 is dedicated to the related work; Section 3, describes our proposed method and the architecture of the framework; In sections 4 and 5, we present our contributions on checking the possible deviations in the elderly's actual behavior derived from monitored data; Lastly some conclusions and indications for future work are provided in section 6.

2 RELATED WORK

Remote health monitoring is an emerging discipline with strong potential to help the elderly in their daily life [7]. By monitoring the status of the person (e.g., heart rate, physical activity, location) via physiological sensors, movement detector, videos, and etc. [27], it is possible to recognize any change in activity as a deviation from their daily life routine and encourage the elderly toward healthy behavior. For healthcare professionals, it is significant to determine the accurate status of the health of a remotely located patient or an aged person, so that, when there is a deviation, appropriate treatment is vetted in a timely manner. Addressing the anomalies in the user behavior includes model the "normal" user behavior based on their daily life routines, recognize the occupancy's behavior in the context and subsequently detect changes from the usual behavior (profiling strategy in [4]). The two leading general taxonomies for classifying, and modeling human behavior are Generic Error Modeling System (GEMS) [28] and phenotypes of erroneous action [13]. GEMS categorized the possible human behavior mistakes in 3 types: rule-based, knowledge-based and slip based mistakes. If the user is aware of the problem, we can bring in the play the rule-based and knowledge-based performance but, if the user has the knowledge about how to perform the activities, slips are supposed to occur, and such analysis can be supported by a "task model".

Indeed, Task analytic methods can be used to describe normative human behavior [15]. Such models (e.g., GOMS [12], AMBOSS [11] , EOFM [6], HAMSTERS [20], ConcurTaskTrees [26]) can show the mental and physical activities which users perform to reach their goals. These models are often hierarchical: activities decompose into other activities and, at the lowest level, atomic actions. Task models have also proven to be of great help for designing and assessing the training program [19], generation of scenarios to be tested over the applications [8], identify possible usability issues [25], find and correct human factors issues in automated systems and generate erroneous human behavior that is a factor in the failure of complex, safety-critical systems [5]. Recent works present approaches and solutions to detect the anomalous behavior of the user by modeling the user behavior. Monekosso and Remagnino [23], described a model-based behavior analysis system for assisted living in which the behavior is defined as any pattern in a sequence of observations and the models are generated from sensed data. A model-based approach is employed for the detection of deviation from the expected behavior: given a model of the system, the predicted output generated by the model is compared to the actual output, and any difference is a potential failure. While in our research, the identified activities (i.e. cooking, eating, etc.) corresponding to the events gathered by sensors are contributed by the caregivers and thus susceptible to errors. Meanwhile, to precisely indicate the nature of the detected anomalies, they used a further examination of a domain expert, unlike, in our work the anomaly detection algorithm detects the anomalous event along with its anomaly classification. There are different methods to find the anomaly based on the user behavior model. In Yi et al., they used Markovian Task Model that aims to model the relationship and dependencies between several separate actions that build a complex activity. In their work, they evaluate their approach for human task recognition by performing peanut butter and jelly sandwich experiment. In this experiment, Yi and Ballard track the eye movement to discover the object on which the eye is fixated and from that to infer the performed subtask [30]. Jakkula et al., [14] describe a method to determine if anomalies can be effectively detected in smart home data using temporal data mining. They refine Allen's temporal predicates [1, 2] to specify relationships between time intervals for use in analyzing smart environment data, and apply it to the task of anomaly detection. They believe anomaly detection is most accurate when it is based on behaviors that are frequent and predictable. While they use machine learning to recognize frequent patterns, in our work we exploit task models specified with the help of relevant stakeholders in order to describe the expected behavior because it is not guaranteed that frequent user patterns represent

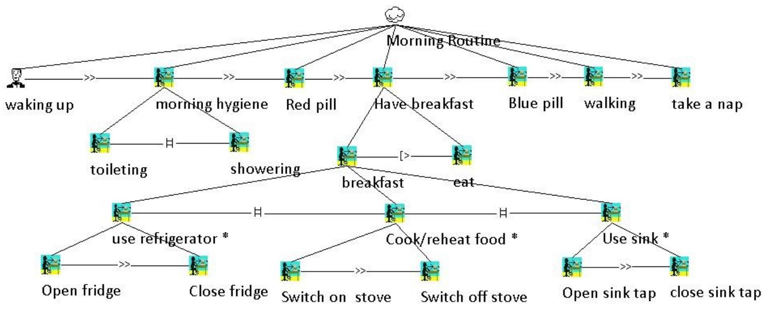


Fig. 1. The task model of the example.

the correct behavior of elder people. Aran et al., [3] proposed a method to detect the anomalies in elderly daily behavior by monitoring the location and the outing of the user. By defining an abstract layer they create a common ground for different sensor configuration. Their approach uses cross-entropy to measure the accuracy of the predicted behavioral changes. basically, by using k-mean clustering they discover the common behavior pattern and by comparing the sequence of event gathered through the sensors to the user's past behavior they define the anomalies. For the evaluation of their proposed method, they used one user written diary as ground truth and compared it with the sequences of events estimated by the abstraction layer. However, they did not investigate the anomaly type, while, in our work, we indicate different types of anomalies at the time of detection. Meng et al., [21] by using the activation status of the sensor and by applying the probabilistic model on the two-layer hierarchy with daily activities modeled the detailed activity of the user and send it to the dynamic daily habit modeling and also to the anomaly detection module. If the similarity between the detected activity and the most similar period in the hierarchy does not reach a specific threshold, the detected activity will be defined as an anomaly and an alarm could be sent to the user. To evaluate the performance of their proposed method, they used two public datasets of fall detection and the Opportunity activity recognition. Although, they are not able yet to precisely detect the complex activities which trigger the activation of the same set of sensors and, as in Aran et al., work [3], they do not distinguish between the different degree of anomalies. More recently, researchers have suggested Machine Learning approaches that most of them require all the data to have accumulated before anomalies can be identified which by modeling the expected user behavior using the task model, we have beaten this problem. Although, obstacles to achieving anomaly detection in real-time include the large volume of data associated with user behavior and the nature of that data [16]. Using our method, we overcome this obstacle by subscribing to the Context Manager in order to be notified when the events related to the user daily activity occur in the context. This way, we just receive the relevant events from the Context Manager.

3 PROPOSED METHOD

The framework for elderly monitoring is based on a task model that describes the planned activities from the user viewpoint (waking up, having breakfast, taking medicine, etc.). However, this task model should be created with the collaboration of someone who has an intimate knowledge of the elderly needs (e.g., caregiver, family member or even the elderly themselves). In particular, the task model includes the *elementary tasks* (i.e., the leaves in the task model tree, which describe the basic activities) that can be associated to one or more events gathered in the context. For highlighting the importance of our task-related anomalies classification, we bring here an example scenario that is written as a short story (by the help of the professional psychologist working at the Sunnaas

Rehabilitation Hospital in Norway) portraying the usual expected morning routine of an elderly. The task model of this scenario is shown in Fig. 1. Sarah is 74 years old, widow and is living alone in her home without any assistance. Her morning routine starts with waking up, washing and getting dressed sometime between 7 and 8 AM. She takes a Red pill with the empty stomach. About half an hour later she prepares and eats breakfast between 8:30- 9:00 (which decomposed several activities such as: using refrigerator, gas, microwave, sink, etc.). After having her breakfast, she takes the Blue pill (which requires having a full stomach). Then, she goes for her daily walk and comes back home before 11:00 and then takes a nap till 11:30.

From this scenario, it is possible to derive different scenarios which Sarah can perform her tasks in a different manner from the expected one. In particular, since the activities can be described concerning a set of task attributes (e.g., temporal relationships, location, time), it is possible to identify a number of corresponding types of task-related anomalies affecting elderly people like i) unusually frequent activities (e.g., too many visits to the toilet); ii) violations of task order relationship (taking medicine before having a meal); iii) missing task (the user forgot to turn off the gas). Based on this model, the framework operates in three steps. In the first step, it logs the events occurring in the context where the elderly actually live, and it associates them with timestamps needed for further analysis. As we will see in the next section describing the architecture of the system, there is a Deviation Analysis module, which subscribes to the Context Manager to receive such events. The events considered by the Deviation Analysis module will not be all the possible events that could be gathered in the current context, but only the ones which are related to the elementary tasks contained in the task model. In the second step, the framework detects the anomalies in user behavior (by using an Anomaly Detection Algorithm), by comparing the expected user behavior (specified in the task model) with the actual user behavior. Several types of deviations can be identified as a result of this comparison (i.e., Less, More, Difference which will be more specifically described in the section 4). Thereby, for the purpose of improving the ability of the framework to detect anomalies, we propose in this paper the novel Anomaly Detection Algorithm. In the third step, the framework uses the deviation detected, the criticality of the task with anomaly and the context of the user to undertake the best intervention action to motivate the users to improve their behavior towards a better lifestyle.

3.1 Preview of the Architecture

Fig. 2 shows the architecture of the framework that we have designed to support the proposed method. Its behavior can be described according to two main phases: one is devoted to configuring the analysis and modeling the user behavior. This phase consists of building/selecting the task model and passing it to the association tool which is a configuration component used to create an association file. The association file is created to store the mappings between the elementary tasks and the events in the context model. The other phase executes the real-time analysis of the incoming events and implements the consequent actions. This phase is centered on the Behavior Analysis module and composed of four sub-modules: Task Model Simulator, Deviation Analysis, Persuasion Module, and the Action Generator. The Deviation Analysis subscribes to the Context Manager in order to be notified each time that an event (related to user daily activity and specified through the association tool) occurs in the context. Also, the Task Model Simulator by referring to the temporal relationships between tasks in task model provides the Deviation Analysis with all the tasks that are logically enabled at any time. So, each time the Deviation Analysis receives an event from Context Manager, it checks out whether the tasks corresponding to the current event happening in the context are logically enabled according to the Task Model Simulator. If not, the Anomaly Detection Algorithm in Deviation Analysis detects the deviation and determines to which category it belongs (this process is repeated for each new event). Consequently, the Deviation Analysis sends

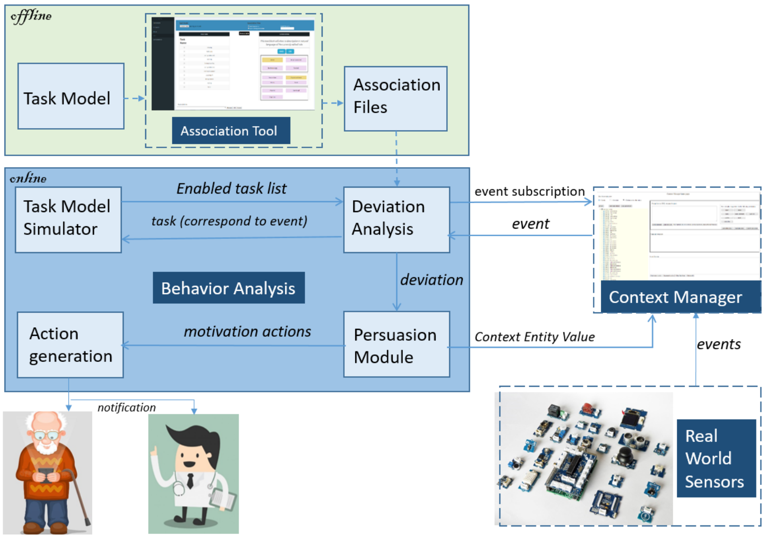


Fig. 2. The Architecture of the Solution.

the deviation (i.e., the triggered event with its degree of an anomaly) to the Persuasion Module. In turn, the Persuasion Module selects the appropriate motivation actions and passes them to the Action Generation module that delivers the appropriate suggestions/indications to the user.

3.2 Task model

The Task Models considered in this approach are stated according to the ConcurTaskTrees (CTT) language [24]and are defined in terms of a hierarchical composition of tasks connected by various temporal operators that describe the temporal relationships among tasks (i.e., enabling (>>), disabling ([>), interruption (| >), choice ([ ]), iteration ($[\overset{*}{t}]$), concurrency (|||) , optionality ([t]) and order independency (| = |)). We consider a task model as composed of n tasks such as: $\Gamma = \{t_1, t_2, \ldots, t_n\}$ which define the expected user behavior. A task t is an activity that should be performed in order to reach a goal and can range from a very high abstraction level (such as deciding a way to prepare a food) to a concrete, action-oriented level (such as "open the fridge door") which we named the latter *elementary tasks* (i.e., the leaves in the task model tree). In our method, a task can be represented formally by a 6-tuple, $t = <N, \iota, \theta, \xi, \tau_s, \tau_f>$, where: N is a task name, ι is a number, defines the maximum number of iteration of the task (which is set by the caretaker or family member of the elderly), θ is a boolean show the task optionality, ξ defines the criticality level of the task consists of {low, medium, high} and $[\tau_s, \tau_f]$ indicate the time interval in which the task may occur. Given a set of tasks in a task model, the elementary tasks represent events that may occur in many different orders according to the temporal operation between them.

Let Σ be the set of all possible sequence of elementary tasks permissible by the CTT task model. Given an $E \in \Sigma$, $E =< t_1, \ldots, t_m > where\ \forall\ i \in [1, m], t_i \ \widehat{=}\ t_j$ for some $j \in [1, n]$. And, let P be a sequence of events received from the user context as $P = <\varepsilon_1, \ldots, \varepsilon_h> where\ \forall\ k \in [1, h],\ \varepsilon_k \ \widehat{=}\ t_i$ for some $i \in [1, m]$, which means each ε in received sequence p corresponds to an elementary task in the task model which in turn corresponds to one or multiple data logged from the real context. Thus, from now on, when we use the ε, it means the event(s) which correspond to a task in the

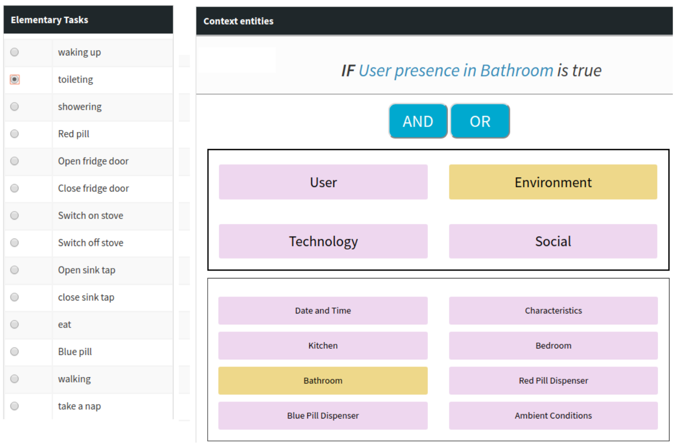


Fig. 3. The association tool

task model and they have the same name. Meanwhile, we used binary symbol ($\circ$) to combine two strings. For example, if $P =< A, B >$ and $p' =< C, D >$, $P \circ P' =< A, B, C, D >$.

Definition 3.1. In the task model we can define S and F as finite, non-empty Arrays of start and final elementary task(s). As $S = \{t_1, ..., t_s\}$ and $F = \{t_f, ..., t_m\}$ for some $1 \leq s < f \leq m$.

Between each task in both the start (S) and the final (F) arrays, there is a choice [ ] temporal operation which specifies two enabled tasks such that, once one has started the other one is no longer enabled. Refer to the Fig. 1, "waking up" is the start task and "take a nap" is the final task in our scenario.

3.3 Association tool

The association tool receives as an input the elderly task model and as an output provides an association file containing the mappings between the elementary tasks (the left panel in the Fig. 3) in the task model and the event in the context model (see the right panel in the Fig. 3). This is a solution for finding the target activities. The context model contains the context entities based on the user context and available sensors. Meanwhile, the customization for a specific user requires only some refinements in the context model. It is worth pointing out, not all the tasks correspond to a single entry (e.g., the "take pill" task corresponds to the event: "open pill" dispenser). One task can have information from multiple entries and be described by a sequence of events (e.g., "having breakfast" task depending on the available sensors can correspond to multiple events: user posture= sit down AND kitchen lamp = on AND oven sensor=on). Therefore, a task will be considered complete if and only if, all the associated events occur. So, effective processing and selection of meaningful events in the context model are necessary to have a proper format for later input to the Deviation Analysis Module.

3.4 Context Manager

The Context Manager (CM) [10] has a client-server architecture. It is a software module composed of a number of context delegates (running on one or more devices located in the environment where the elder lives) and a context server. Context delegates take information from sensors and pass it on to the Context server, which organizes the data into a common vocabulary used for communication with other platform components. The main goals of this subsystem are detecting the current values

```
{events:[{
name:"Sarah/morning_routin/Red pill",
ref:"/Sarah/Environment/@Dispenserpill",
value:"open",
verified:"true"
]}
Or
{events:[{
name:"Sarah/morning_routin/Red pill",
ref:"/Sarah/Environment/@Dispenserpill",
value:"open",
verified:"false"
]}
```

Listing 1. Notification received from CM.

of a wide range of variables (i.e., related to the user, environment, technology) and informing other architectural modules about relevant changes in such values. Asynchronous notifications are automatically sent by the context server to modules that have previously subscribed (in our case, Deviation Analysis) for a particular state or for changing one or more parameters. The advantage of the asynchronous approach is that the subscriber module is not required to continuously query the context server for the current values of the involved parameters. As we mentioned before, P is a sequence of events (coincide with the data logged from the real context) received from the CM at the current time. Thus, once the event(s) specified in the association file occur in the current context, which means the Context Manager determines that the current event is one of those requested from the subscriber (i.e., the Deviation Analysis), the *CM* sends a JSON format notification (e.g., Listing 1) to the Deviation Analysis. Each notification received from the context manager may consist of one or multiple events which altogether correspond to one task in the task model. So, if in the association tool the mapping between the elementary task and the event in the context model is 1:1, the context manager sends a notification with one event and if the mapping is 1: N, the CM sends a notification contains multiple events right after the last event occurs.

Each event ε is expressed as 5-tuple , $< \mathcal{N}, \mathcal{R}, \gamma, \upsilon, \tau >$ where $\mathcal{N}$ ("name" in Listing1), is a string composed of the user name, associated file name and the elementary task name associated with the context entity. In the following sections ε always corresponds to the elementary task name in the task model (*Redpill* in Listing 1) which is exactly the name of the task t_1 (see *Redpill* in Fig. 1) in the task model. Next element, $\mathcal{R}$ indicates the event path in the context model (see "ref" in Listing1), γ is a value which has been set when we mapped between elementary task and the context entity in association tool (see "value" in Listing1), υ determine whether the change appear in γ was verified (see "verified" in Listing1) and finally, τ defines the timestamp of event.

3.5 Task Model Simulator

We implemented a Task Model Simulator, which gets as an input the elderly task model, starts from the first elementary task and hands out the list of tasks currently enabled (based on the temporal operators between the tasks). This process continues till it arrives at the final task in the task model. Indeed, the Task Model Simulator acts as a function $f : t \mapsto EN_t$, which is a map where t is the currently executed elementary task and EN_t is the array of enabled tasks after the executed task. So, for the events in the sequence received from the context manager: $\varepsilon_2 \in EN_{\varepsilon_1} \iff \exists \ P \in \Sigma : P =< P_1 \circ \varepsilon_1 \circ \varepsilon_2 \circ P_2 >$. Actually, the main goal of the Simulator module is

```
<op>subscribe_event</op>
<subscriber_address>http://x.isti.cnr.it/ae_http</subscriber_addres>
<event>
  <simple_event event_name="Red pill" xPath="/user/environment/
      @Dispensrpill"/>
</event>
<condition operator="eq">
  <entityReference xPath="/user/Sarah/environment/@Dispensrpill"/>
  <constant value="open" type="bool"/>
</condition>
```

Listing 2. Example of Subscribtion to the Context Manager.

to detect if the task corresponding to the event(s) in partial sequence (p) detected by the context manager is one of those enabled according to the temporal relationships between tasks in task model.

3.6 Deviation Analysis

The Deviation Analysis module receives three elements as input: i) the association file; ii) the event lastly occurred in the current context (from the Context Manager); iii) the enabled task sets (produced by the Task Model Simulator). The module subscribes the association file to the Context Manager in order to be notified when the events specified in the association file occur in the context. In the Listing 2, you can see an example of an XML message that a Deviation Analysis module sends to the Context Manager to subscribe for specific medicine intake event. Thus, as soon as an event (specified in the association file) occurs, the Context Manager sends a notification to the Deviation Analysis. For example in Listing 1, when the Context Manager determines that the current value of medicine dispenser is equal to open, immediately notifies the Deviation Analysis module. Afterward, the Deviation Analysis extracts the event name, value, and timestamp from the CM notification, it finds the corresponding elementary task and sends these data to the Anomaly Detection Algorithm. The algorithm checks whether the order and the time of the detected event correspond to the user expected behavior sequence (defined in task model). If not, the algorithm specifies the time and the type of the anomaly we encounter.

3.7 Persuasion Module

The goal of the Persuasion Module, is using the detected deviation received from the Deviation Analysis, the critical level of the involved event and the user context in order to generate persuasive actions that can be applied to reinforce or change the anomalous behavior of the user. Depending on the deviation type and the user context, actions can be different such as *Activate some functionality*; *Generate alarms* (to highlight some potentially dangerous situations); *Send reminders* (to indicate a task that should have been accomplished); *Provide persuasive suggestions* (to encourage users to change their behavior); *Send prompt, messages, provide explanation messages, change the state of the appliance* (light, fridge,...). Later, the decided action will be sent to the Action Generation Module.

3.8 Action Generator

The Action Generation module by considering relevant contextual aspects, defines the most suitable way to render the actions identified by Persuasion Module for addressing the diagnosed deviations. Thus, based on the current state of the user (i.e., location, disabilities, etc.), it decides where (i.e., user smartphone, tablet, TV) and how (i.e., graphically, text, audio) delivers the action to the elderly.

4 POSSIBLE TASK-RELATED ANOMALIES IN ELDERLY BEHAVIOR

As already mentioned, an anomalous situation is characterized by a dissimilarity between the observed situation (derived from sensors in the current user context) and the expected one (E) described in a task model. The identified deviation could be characterized by different degrees of severity and could be the sign of various situations, the beginning of an unhealthy habit of the elderly, the insurgence of an illness or even a serious physical/mental decline. Thus, analyzing the behavior of the elderly can be useful to highlight trends as well as giving recommendations and health-related reminders to them. In Reason's taxonomy [28], slips (i.e. the user has the knowledge about how to perform the activities) occur due to attention failures and can manifest as omissions, repetitions, and commissions. Our anomalous human behavior generation method, models these slips in a more systematic categorization of the various types of deviations which we have associated with specific keywords. In our method, we considered two types of classification for the different grades of anomalies: i) anomaly in the complete and ii) in the prefix sequences. In the former, anomalies are defined based on the whole sequence at the end of the day and in the latter, the different degree of anomalies are considered for the partial sequences (prefixes) which later will be associated to the one or some types of anomaly in the former classification. The general classification of the anomalies considering the whole sequence produced by the context manager at the end of the day (sequence P) is as follow:

Less: An event that was expected has not been performed. Thus, P is Less and ε is a missing event if:

$$P \notin \Sigma \quad \wedge \quad P =< P_1 \circ P_2 > \wedge \ \exists E \in \Sigma : E =< P_1 \circ \varepsilon \circ P_2 > \tag{1}$$

More: An event has been performed more than expected time. Hence, P is More and ε is an extra event if:

$$P \notin \Sigma \quad \wedge \quad P =< P_1 \circ \varepsilon \circ P_2 > \wedge \ \exists E \in \Sigma : E =< P_1 \circ P_2 > \tag{2}$$

Difference: The tasks considered have been performed differently from what the designer intended in the task model. So, P is Difference and ε is an event out of its order if:

$$\begin{aligned} &P \notin \Sigma \quad \wedge \quad P =< P_1 \circ \varepsilon_1 \circ \varepsilon_2 \circ P_2 > \wedge \varepsilon_2 \notin EN_{\varepsilon_1} \\ &\exists E \in \Sigma : E =< P_1 \circ \varepsilon_1 \circ \varepsilon_3 \circ P_2 > \wedge \varepsilon_3 \neq \varepsilon_2 \end{aligned} \tag{3}$$

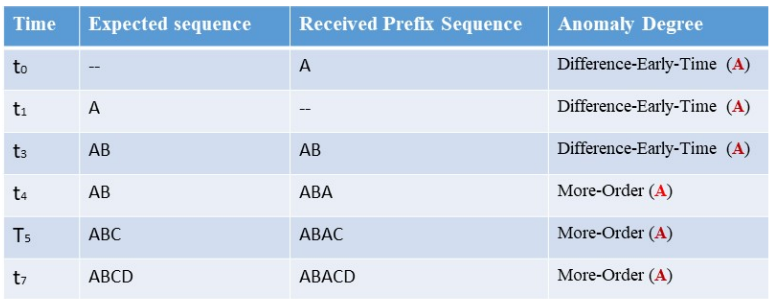
No-Anomaly: An event has none of the above conditions.

$$P \in \Sigma \tag{4}$$

As we mentioned before, in real-time as each event receives from the Context Manager, our partial sequence (prefix) get changed. The anomalies in prefixes can change as the partial sequence get completed over time. As we defined in section 3.2, $E =< t_1, ..., t_m >$ is a sequence of tasks expected from the user and $P =< \varepsilon_1, ..., \varepsilon_h >$ is a partial sequence of events received from the context manager at the current time. Given $h \leq m$, we define prefix $E' = < \varepsilon_1, ..., \varepsilon_h >$ where$\forall i \in [1, h], \varepsilon_i = t_j$, for some$j \in [1, m]$. For finding the anomalies in the prefixes, each time an event receives, we compare prefix P with the prefix E'. It is worth to mention that, the events time are embedded in them and the ascending relationship between events time has been preserved means: $\varepsilon_1 \circ \varepsilon_2 = \varepsilon_1^{t_1} \circ \varepsilon_2^{t_2} \Rightarrow t_2 > t_1$. Given $P \notin \Sigma$, the anomalies in the partial sequences are categorized as:

- ***Less_Partial:*** An event that was expected to be carried out, has not been performed in the partial sequence.

$$\begin{aligned} &p =< P_1 \circ \varepsilon_i > \wedge \exists E \in \Sigma : E =< P_1 \circ \varepsilon_{i-1} \circ \varepsilon_i \circ P_2 > \wedge \\ &\nexists E' \in \Sigma : E' =< P_1 \circ \varepsilon_i \circ P_2 > \text{for any } \ P_2 \wedge \ \ current - time > \varepsilon_{i-1}.\tau_f \end{aligned} \tag{5}$$



Time	Expected sequence	Received Prefix Sequence	Anomaly Degree
t₀	--	A	Difference-Early-Time (A)
t₁	A	--	Difference-Early-Time (A)
t₃	AB	AB	Difference-Early-Time (A)
t₄	AB	ABA	More-Order (A)
T₅	ABC	ABAC	More-Order (A)
t₇	ABCD	ABACD	More-Order (A)

Fig. 4. The example of task-related anomalies

- ***Difference-Early-time:*** An event that was expected to be carried out in certain interval of time, has been performed before its starting time.

$$p =< P_1 \circ \varepsilon^t \circ P_2 > \wedge \exists E' \in \Sigma : E' =< P_1 \circ \varepsilon \circ P_2 > \wedge \ t < \tau_s \tag{6}$$

- ***Difference-Later-Time:*** An event that was expected to be carried out in certain interval of time, has been performed after its completing time is finished.

$$p =< P_1 \circ \varepsilon^t \circ P_2 > \wedge \exists E' \in \Sigma : E' =< P_1 \circ \varepsilon \circ P_2 > \wedge t > \tau_f \tag{7}$$

- ***More-Order:*** An event is performed more times than expected and out of its order.

$$\begin{aligned} &P =< P_1 \circ \varepsilon_i \circ P_2 \circ \varepsilon_j >: \varepsilon_i = \varepsilon_j \ , \varepsilon_j \notin EN_{\varepsilon_{j-1}} \ \wedge \\ &\exists E' \in \Sigma : E' =< P_1 \circ \varepsilon_i \circ P_2 \circ \varepsilon_k > \wedge \varepsilon_k \neq \varepsilon_j \end{aligned} \tag{8}$$

- ***More-Number:*** An event is performed more times than expected in his time interval and order.

$$P =< P_1 \circ \varepsilon_i \circ \varepsilon_j \circ P_2 >: \varepsilon_i = \varepsilon_j \ , \exists E' \in \Sigma : E' =< P_1 \circ \varepsilon_i \circ P_2 > \tag{9}$$

- ***Difference-Order:*** An event has been performed in a wrong temporal order.

$$P =< P_1 \circ \varepsilon_1 \circ \varepsilon_2 \circ P_2 > \wedge \ \varepsilon_2 \notin EN_{\varepsilon_1}, \exists E' \in \Sigma : E' =< P_1 \circ \varepsilon_1 \circ \varepsilon_3 \circ P_2 > \wedge \varepsilon_3 \neq \varepsilon_2 \tag{10}$$

- ***Difference-Order-Time:*** An event considered has been performed in a wrong temporal order and at a wrong time.

$$P =< P_1 \circ \varepsilon^t \circ P_2 \circ P_3 > \wedge \ \exists E' \in \Sigma : E' =< P_1 \circ P_2 \circ \varepsilon \circ P_3 > \wedge \ \wedge \varepsilon.t \notin [\tau_s, \tau_f] \tag{11}$$

As we referred before, as the prefix sequence evolves (with the occurrence of each new observation), the output of the anomaly detection (and thus the potential anomaly already identified) may change or it may remain the same. For example let $E =< A_{(t_1 - t_2)}, B_{(t_3 - t_4)}, C_{(t_5 - t_6)}, D_{(t_7 - t_8)} >$ be a sequence that will be expected from the user where A, B, C, D are the events (correspond to elementary tasks) and $[t_1, t_8]$ is the time interval of the events performance. Thus, given $P =< A_{(t_0)}, B_{(t_3)} >$ as a prefix sequence, we have *Difference-Early-time* anomaly on event *A*. Assume that at t_4 arrives event *A* again. So, our prefix sequence become $P =< A_{(t_0)}, B_{(t_3)}, A_{(t_4)} >$ and subsequently the anomaly type of the event *A* changes to *More-Order* based on equation 8. As you see in Figure 4, an anomaly type of an event (e.g. event *A*) may change over time depends on the event attributes, the time and the order of the event performance.

LEMMA 4.1. *Anomalies in Equation 5-11 are pairwise disjoint. Given P as a prefix sequence, P cannot have two or more classification of the anomaly on the same event at the same time.*

ALGORITHM 1: Anomaly detection

```
Data: event sequence: P = $< \varepsilon_0, \varepsilon_1, ....\varepsilon_{n-1} >$
Result: $\varepsilon$ with its anomaly type
while $i < n$ do
    if $\varepsilon_i.v$ = "true" then
        if $\varepsilon_i.t > \tau_f$ then marks $\varepsilon_i$ as Difference-Later-Time
        if $\varepsilon_i.t < \tau_s$ then marks $\varepsilon_i$ as Difference-Early-Time
        $Dup \leftarrow Delete\_Duplication(P)$ and marks $Dup$ as More
        constract E by using $(P, S, F)$ and marks $Dif$ and $Les$ as Difference and Less
        while $flag == true$ do
            for $\forall i \in [0, n-1]$ do
                $swapcount \leftarrow 0$                // number of time we swap the events
                $F\_error \leftarrow 0$           // number of times we encounter nonswappble events
                if $\varepsilon_i \notin EN_{\varepsilon_{i-1}} \wedge \varepsilon_{i-1} \in EN_{\varepsilon_i}$ then Swap $(\varepsilon_i, \varepsilon_{i-1})$; $swapcount++$
                else $F-error++$
            end
        end
        if $swapcount > 0$ then $Flag = true$ else $Flag = false$
        if $F\_error > 0$ then                     // there exist 2 non-swappable tasks
            $Sp \leftarrow FindShortestPath\ (\varepsilon_{i-1}, \varepsilon_i) \rightarrow E =< P, Sp, \varepsilon_i >$
            $Delete\_Duplication\ (E)$
            $Less \leftarrow |E - P|$
            $difference1 \leftarrow |LCS(P, E) - P|$
            $difference2 \leftarrow |LCS(E, P) - P|$
            $Difference \leftarrow difference1 \cup difference2$
        end
    end
    if $\exists \varepsilon \in$ Less $\wedge\ current - time > \varepsilon.\tau_f$ then $\varepsilon.anomaly \rightarrow$ Less-Partial
    if $\exists \varepsilon \in$ Difference-Later-Time $\cup$ Difference-Early-Time $\wedge\ \varepsilon \in$ Difference then
      $\varepsilon.anomaly \rightarrow$ Difference-Order-Time
    if $\exists \varepsilon_i \in$ Difference, $\varepsilon_i \notin EN_{\varepsilon_{i-1}}$ then $\varepsilon_i.anomaly \rightarrow$ Difference-Order
    if $\exists \varepsilon \in$ More $\wedge\ \varepsilon_i \notin EN_{\varepsilon_{i-1}}$ then $\varepsilon.anomaly \rightarrow$ More-Order else $\varepsilon.anomaly \rightarrow$ More-Number
end
```

Proof. Disjoint anomalies cannot happen at the same time. In other words, they are mutually exclusive. we prove our statement by contradiction. For example, let $< A_{(t_1-t_2)}, B_{(t_3-t_4)}, C_{(t_5-t_6)} >$ be a sequence that is expected from the user. We assume, at t_4 we have a prefix P as $< A_{(t_1)}, B_{(t_2)}, C_{(t_3)} >$ and event B has two kind of anomalies *Difference-Order* (Equation 10) and *Difference-Early-time* (Equation 6) at time t_2. So, based on equation 6 : $B \in EN_A$, $t_2 < t_3$ and based on equation 10 : $B \notin EN_A$, $t_2 \in [t_3, t_4]$, which obviously, $t_2 < t_3$ but $t_2 \notin [t_3, t_4]$ and also, event B can not be enabled by event A and at the same time not be enabled by event A. Thus, event B at t_2 has just *Difference-Early-time* anomaly. The proofs for other cases are straightforward and omitted for brevity.

□

4.1 Anomaly detection Algorithm:

For each received event from the Context Manager, the Anomaly Detection Algorithm 1 starts to construct an expected sequence (E) permissible by CTT task model, using the currently received event and the information generated by the Task Model Simulator. Sequence E is one of the possible sequence of events with respect to the temporal operators between tasks in the user task model and serves as a ground truth to be compared with the current received event sequence (P) for finding the anomalies.

So, first the algorithm checks the interval time of the received event (line 3, 4), if the equation 6 holds, marks it as *Difference-Early-Time* and if the equation 7 holds, marks the event as *Difference-Later-Time*. Then, in line 5 controls if in the partial sequence P, there is any event equal to the currently received event, if yes, controls the maximum allowed repeated number of event, if the threshold has been violated marks the received event as *More* and deletes it from the sequence P. As we have already mentioned, our algorithm makes the expected sequence E step by step. As the first step, at line 6 the algorithm finds the start and the final events in the task model and inserts/shifts them at the start and the end position in the sequence E as $E =< S, P, F >$. Therefore, the algorithm controls whether the event sequence P includes any member of the start (S) and the final (F) events in the task model. If any member in S and F exists in P but not at the start and the end of the P (see D^s and D^f in Table 1), it shifts them to the start and the final position in the E and marks these events (see *Dif* in Table 1) as *Difference*. While, if any member of S or F do not exist in P (see L^s and L^f in Table 1), it inserts the start and the final event(s) at the start and the end of the sequence E and marks these events (see *Les* equation in Table 1) as *Less*. Further, the algorithm (line 7-14) checks if the currently received event corresponds to an enabled event of the last event in P. If so, the received event was an expected one (according to the task model), then the process iterates in a similar manner for the next event in the received event sequence. Otherwise (i.e., Equation 3), it controls the opposite order to see if the previously executed event corresponds to an enabled event of the currently executed event. If yes, it swaps two consecutive events (which are not in the right order) for the sake of making the expected sequence (E). While, if none of the above conditions were applicable, it means there are missing event(s) between the currently received event and the last event in partial sequence P. For finding these missing events (line 17), the function *FindShortestPath()* takes in input these two successive events (that are not swappable) and finds the shortest paths between them. Then, it fills up the distance between these two events by the missing events and deletes the repetitive events (line 18) in the new sequence if there is any (for the sake of making expected sequence E). Later, by comparing the received event sequence P and the expected event sequence E, it finds any missing events and marks them as *Less* (line 19). Now, we have an expected sequence E. Next, the Longest Common subsequence *(LCS)* function finds the longest subsequence common between two sequences (E, P). It compares the result of the *LCS* with P and saves the output (if it is any) as *difference1*. Later, it repeats this process by switching the parameters as $LCS(P, E)$ and compares the result with the partial sequence P and saves the output as *difference2* (line 20-21). Later,at line 22, it merges the result of *difference1* and *difference2* and mark them as *Difference*.

The last step is announcing the anomaly of the received event sequence (line 25- 28). In this step, the algorithm checks first the events marked as *Less* , if the current time is grater than their time interval (τ_f) , the algorithm saves these events as a *Less-Partial* anomaly. Otherwise, they remain marked as *Temporary-Less* for the future analysis. If the tasks which have been marked as *Difference-Early-Time* or *Difference-Later-Time* are even marked as *Difference*, their anomaly becomes *Difference-Order-Time* , on the other hand, they remain *Difference-Early-Time* and *Difference-Later-Time* . If the events which marked as *More* are not enabled by their previous event (Equation 10),

D^s	$= \left\{\varepsilon_i \in P : \varepsilon_i \in S \wedge \exists \;\; \varepsilon_j \notin S, \varepsilon_j \in P \;\; j < i\right\}$
D^f	$= \left\{\varepsilon_i \in P : \varepsilon_i \in F \wedge \exists \;\; \varepsilon_j \notin F, \varepsilon_j \in P \;\; j > i\right\}$
Dif	$= D^s \cup D^f$
L^s	$= \left\{\varepsilon_i \in S : \varepsilon_i \notin P\right\}$
L^f	$= \left\{\varepsilon_i \in F : \varepsilon_i \notin P\right\}$
Les	$= L^s \cup L^f$

Table 1. Set the start and the final task.

announces them as *More-Order* , otherwise, they are *More-Number* . And finally, for the events marked as *Difference*, the algorithm checks again the equation 10 and if its true, it announces these events as *Difference-Order.*

5 EVALUATION

We developed a tool to detect the deviations by comparing the expected user behavior (specified in the Task Model) with the real one. In order to have a systematic analysis of the tool ability to detect the anomalies, we carried out a laboratory evaluation. In favor of systematically testing the Anomaly Detection Algorithm, we decided to simulate the event notifications received from the Context Manager. Simulating the events detected by the Context Manager facilitates understanding the system's behavior without actually testing the system in the real world and help us to meet the experiment objectives. First, we obtained 10000 sequences (Σ) of events which are permissible in the CTT task model described the expected elderly behavior. Next, we generated the anomalous sequences (Σ_1) obtained by manipulating the Σ and applying one of the anomalies described in section 4. Then, we created other anomalous sequences (Σ_2) that have more than one anomaly type at different times, obtained by manipulating the previously generated anomalous sequences in (Σ_1). Finally, the Context Manager Simulation sent the events from the Σ, Σ_1 and Σ_2, one by one, to the Deviation Analysis module and the Detecting Anomaly Algorithm as a result reports the detected abnormal events with their anomaly classification, and the corresponding accuracy was assessed.

5.1 Context Manager Simulation

As mentioned earlier, the first step is simulating the event notifications received from the Context Manager. To this aim, the simulator takes all the elementary tasks in the task model, associate them with the data necessary to execute the corresponding event (e.g., time, location, value) as in listing 1. Second, for making the initial sequences Σ, the simulator selects randomly an event from those corresponding to the initially enabled set of events according to the task model. Then, it randomly chooses an event between all the enabled events after the first event. This process continues until it arrives at the one of the enabled final events. Meanwhile, the duplicated sequences are deleted. The next step consists of generating the sequences Σ_1 with only one type of anomaly. When creating Σ_1 the algorithm considers all types of anomaly classifications. So, given $E \in \Sigma$ and $E_1 \in \Sigma_1$, the algorithm generates the anomalous sequences as follow:

(1) *Less*: Randomly omits an event from each sequence in Σ.
Given $E \in \Sigma : E =< P_1 \circ \varepsilon_1 \circ P_2 > \Rightarrow E_1 \in \Sigma_1 : E_1 =< P_1 \circ P_2 >$.

(2) *Difference-Early-time*: Randomly chooses an event from each sequence in Σ and anticipates event time one minute before its starting time (τ_s). The change is made in such a way to still respect the events order.
Given $E \in \Sigma : E =< P_1 \circ \varepsilon_1^{t_1} \circ \varepsilon_2^{t_2} \circ P_2 > \Rightarrow E_1 \in \Sigma_1 : E_1 =< P_1 \circ \varepsilon_1^{t_1} \circ \varepsilon_2^{t_2-1} \circ P_2 >$.

Measure	Rate
Sensitivity	99%
Specificity	98%
FPR	2%
Accuracy	95%

Table 2. Algorithm performance summary

(3) *Difference-Later-time*: Randomly chooses an event from each sequence in Σ and brings the event time one minute after its final time (τ_f) with respect to the events order.
Given $E \in \Sigma : E =< P_1 \circ \varepsilon_1^{t_1} \circ \varepsilon_2^{t_2} \circ P_2 > \Rightarrow E_1 \in \Sigma_1 : E_1 =< P_1 \circ \varepsilon_1^{t_1} \circ \varepsilon_2^{t_2+1} \circ P_2 >$.
(4) *Difference-Time-order*: Randomly chooses an event from each sequence in Σ and changes the order and the time of the event.
Given $E \in \Sigma : E =< P_1 \circ \varepsilon^{t_2} \circ P_2 > \Rightarrow E_1 \in \Sigma_1 : E_1 =< P_1 \circ P_2 \circ \varepsilon^{t_3} >, t_3 \notin [\tau_s, \tau_f]$.
(5) *Difference-Order*: Randomly chooses an event from each sequence in Σ and changes its order with respect to the sequence Σ in a way that the event time remains in its time interval.
Given $E \in \Sigma : E =< P_1 \circ \varepsilon_i^{t_1} \circ \varepsilon_j^{t_2} \circ P_2 > \Rightarrow E_1 \in \Sigma_1 : E_1 =< P_1 \circ \varepsilon_i^{t_1} \circ P_2 \circ \varepsilon_j^{t_2} >$
(6) *More-number*: Chooses randomly one event from each sequence in the Σ and creates another instance of the same event that is located in a position allowed by the task model.
Given $E \in \Sigma : E =< P_1 \circ \varepsilon_i^{t} \circ P_2 > \Rightarrow E_1 \in \Sigma_1 : E_1 =< P_1 \circ \varepsilon_i^{t_1} \circ \varepsilon_i^{t_2} \circ P_2 >, t \in [t_1, t_3]$.
(7) *More-order*: Choose randomly one event from each sequence in Σ and creates another instance of the same event that is located in a position not allowed by the task model.
Given $E \in \Sigma : E =< P_1 \circ \varepsilon_i \circ P_2 > \Rightarrow E_1 \in \Sigma_1 : E_1 =< P_1 \circ \varepsilon_i \circ P_2 \circ \varepsilon_i >$.

Later, to show that the algorithm is capable of detecting different types of anomalies that occur at different times, the simulator introduces another type of anomaly, thus obtaining sequences containing two or more types of anomalies (Σ_2). For this aim, the simulator uses the generated anomalous sequences in Σ_1, chooses the anomalous event from each sequence and duplicates, omits or changes the event order and time to generate another type of anomalous sequences Σ_2. Finally, for validating the algorithm, we fed the Deviation Analysis module by the events in the sequences Σ, Σ_1 and Σ_2. The anomaly detection Algorithm detects the anomalies and as an output it provides the event name with its anomaly type and the time when the anomaly has been detected.

5.2 Results

The experimental results of our Anomaly Detection Algorithm are shown in Table 2. The performance of the proposed system is shown in Table 3. We measured sensitivity and specificity to evaluate algorithm performance. Sensitivity is defined as the capacity of the system for correctly identifying true abnormal events. While, specificity is defined as the capacity of the system for not generating false positives. These values were calculated as in equation 12 and 13 :

$$Sensitivity = \frac{TP}{FN + TP} \tag{12}$$

$$Specificity = \frac{TN}{FP + TN} \tag{13}$$

Actually, Positive/Negative means that the model predicts that the sequence corresponds to abnormal/normal and True/False means that the prediction is right/wrong. True positives (TP) are the number of case where abnormal sequences are correctly detected and classified by the system.

		System detection	
		Abnormal	Normal
simulation	Abnormal	$TruePositive$: 18850	$FalsePositive$: 150
	Normal	$FalseNegative$: 100	$TrueNegative$: 9900

Table 3. Experimental results with the simulation

False negatives (FN) are the number of cases where normal sequences classified as abnormal. True negatives (TN) are the number of cases where normal sequences are correctly detected by the algorithm. False positives (FP) are the number of cases where abnormal sequences are falsely detected as normal. Another important measure is False Positive Rate (FPR), which is defined as the number of false positive instances (FP) divided by the number of False Positives (FP) and True Negatives (TN). A hight FPR can significantly decrease the utility of the anomaly detection system. Finally the accuracy of the algorithm is calculated as:

$$ACCURACY = \frac{TP + TN}{TP + TN + FP + FN} \tag{14}$$

The experimental results, showed that our online Anomaly detection Algorithm has the accuracy of 95% and a false positive rate of 2%. The algorithm detects the event that triggered the anomaly and indicates the anomaly type (Equation 1- 11) and when it occurs.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we present a method to detect abnormal elderly behavior in real-time, which is important in AAL scenarios because these anomalies may represent early signs of health-related issues. We have developed a profiling strategy in which task models specify the "normal" behavior, which can also work in case of rare anomaly data. Our solution compares the expected behavior expressed in such models with the sequence of events generated by the real user. For achieving this result, we had to simulate the event sequences received from the Context Manager (which represent the user daily activities). Then, we validated the algorithm performance by sending the simulated normal and anomalous sequences to the Deviation Analysis module. The experiment results are promising. For future work, we plan to test the algorithm with data gathered from real users and act upon the detected deviations in a persuasive manner aiming to help the elderly well-being (Section 3.7). For this reason, we aim to use behavioral change techniques which are theory-driven methods and are used in behavior change interventions to define the best motivation action in Persuasion Module. These techniques can be defined as coordinated sets of activities designed to change specified behavior patterns. There is a large number of interventions described in the literature and have been ongoing efforts to categorize such interventions into taxonomies [22]. Lastly, the currently proposed method is convenient for the solitary elderly. Further research to detect the anomalies based on user activities performed by multiple users is in our interest as well.

REFERENCES

[1] James F Allen. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26, 11 (1983), 832–843.

[2] James F Allen and George Ferguson. 1994. Actions and events in interval temporal logic. *Journal of logic and computation* 4, 5 (1994), 531–579.

[3] Oya Aran, Dairazalia Sanchez-Cortes, Minh-Tri Do, and Daniel Gatica-Perez. 2016. Anomaly detection in elderly daily behavior in ambient sensing environments. In *International Workshop on Human Behavior Understanding*. Springer, 51–67.

[4] UABUA Bakar, Hemant Ghayvat, SF Hasanm, and SC Mukhopadhyay. 2016. Activity and anomaly detection in smart home: A survey. In *Next Generation Sensors and Systems*. Springer, 191–220.

[5] Matthew L Bolton and Ellen J Bass. 2013. Generating erroneous human behavior from strategic knowledge in task models and evaluating its impact on system safety with model checking. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43, 6 (2013), 1314–1327.

[6] Matthew L Bolton, Radu I Siminiceanu, and Ellen J Bass. 2011. A systematic approach to model checking human–automation interaction using task analytic models. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 41, 5 (2011), 961–976.

[7] Zoraida Callejas and Ramón López-Cózar. 2009. Designing smart home interfaces for the elderly. *ACM SIGACCESS Accessibility and Computing* 95 (2009), 10–16.

[8] José Creissac Campos, Camille Fayollas, Marcelo Gonçalves, Célia Martinie, David Navarre, Philippe Palanque, and Miguel Pinto. 2017. A More Intelligent Test Case Generation Approach through Task Models Manipulation. *Proceedings of the ACM on Human-Computer Interaction* 1, 1 (2017), 9.

[9] Céline Franco, Jacques Demongeot, Christophe Villemazet, and Nicolas Vuillerme. 2010. Behavioral telemonitoring of the elderly at home: Detection of nycthemeral rhythms drifts from location data. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*. IEEE, 759–766.

[10] Giuseppe Ghiani, Marco Manca, Fabio Paternò, and Carmen Santoro. 2017. Personalization of context-dependent applications through trigger-action rules. *ACM Transactions on Computer-Human Interaction (TOCHI)* 24, 2 (2017), 14.

[11] Matthias Giese, Tomasz Mistrzyk, Andreas Pfau, Gerd Szwillus, and Michael Von Detten. 2008. AMBOSS: a task modeling approach for safety-critical systems. In *Engineering Interactive Systems*. Springer, 98–109.

[12] M Helander. 1988. Towards a practical GOMS model methodology for user interface design. In *Handbook of human-computer interaction*. North-Holland, 135–158.

[13] Erik Hollnagel. 1993. The phenotype of erroneous actions. *International Journal of Man-Machine Studies* 39, 1 (1993), 1–32.

[14] Vikramaditya Jakkula, Diane J Cook, et al. 2008. Anomaly detection using temporal data mining in a smart home environment. *Methods of information in medicine* 47, 1 (2008), 70–75.

[15] Barry Kirwan and Les K Ainsworth. 1992. *A guide to task analysis: the task analysis working group*. CRC press.

[16] Linda Lankewicz and Mark Benard. 1991. Real-time anomaly detection using a nonparametric pattern recognition approach. In *Computer Security Applications Conference, 1991. Proceedings., Seventh Annual*. IEEE, 80–89.

[17] Ahmad Lotfi, Caroline Langensiepen, Sawsan M Mahmoud, and Mohammad Javad Akhlaghinia. 2012. Smart homes for the elderly dementia sufferers: identification and prediction of abnormal behaviour. *Journal of ambient intelligence and humanized computing* 3, 3 (2012), 205–218.

[18] Marco Manca, Parvaneh Parvin, Fabio Paternò, and Carmen Santoro. 2017. Detecting anomalous elderly behaviour in ambient assisted living. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 63–68.

[19] Célia Martinie, Philippe Palanque, David Navarre, Marco Winckler, and Erwann Poupart. 2011. Model-based training: an approach supporting operability of critical interactive systems. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 53–62.

[20] Célia Martinie, Philippe Palanque, and Marco Winckler. 2011. Structuring and composition mechanisms to address scalability issues in task models. In *IFIP Conference on Human-Computer Interaction*. Springer, 589–609.

[21] Lei Meng, Chunyan Miao, and Cyril Leung. 2017. Towards online and personalized daily activity recognition, habit modeling, and anomaly detection for the solitary elderly through unobtrusive sensing. *Multimedia Tools and Applications* 76, 8 (01 Apr 2017), 10779–10799. https://doi.org/10.1007/s11042-016-3267-8

[22] Susan Michie, Maartje M van Stralen, and Robert West. 2011. The behaviour change wheel: a new method for characterising and designing behaviour change interventions. *Implementation science* 6, 1 (2011), 42.

[23] Dorothy N Monekosso and Paolo Remagnino. 2010. Behavior analysis for assisted living. *IEEE Transactions on Automation science and Engineering* 7, 4 (2010), 879–886.

[24] Giulio Mori, Fabio Paternò, and Carmen Santoro. 2002. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transactions on software engineering* 28, 8 (2002), 797–813.

[25] Laila Paganelli and Fabio Paternò. 2003. Tools for remote usability evaluation of Web applications through browser logs and task models. *Behavior Research Methods* 35, 3 (2003), 369–378.

[26] Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. 1997. ConcurTaskTrees: A diagrammatic notation for specifng task models. In *Human-Computer Interaction INTERACT'97*. Springer, 362–369.

[27] François Portet, Michel Vacher, Caroline Golanski, Camille Roux, and Brigitte Meillon. 2013. Design and evaluation of a smart home voice interface for the elderly: acceptability and objection aspects. *Personal and Ubiquitous Computing* 17, 1 (2013), 127–144.

[28] James Reason. 1990. *Human error*. Cambridge university press.

[29] Janine L Wiles, Annette Leibing, Nancy Guberman, Jeanne Reeve, and Ruth ES Allen. 2012. The meaning of âĂIJaging in placeâĂİ to older people. *The gerontologist* 52, 3 (2012), 357–366.

[30] Weilie Yi and Dana H Ballard. 2006. Behavior recognition in human object interactions with a task model. In *Video and Signal Based Surveillance, 2006. AVSS'06. IEEE International Conference on*. IEEE, 64–64.

Received M YYYY; revised YYY; accepted M YYYY