






Supporting Users in the Continuous Evolution of Automated Routines in Their Smart Spaces

Estefanía Serral¹, Daniel Schuster², and Yannis Bertrand¹

¹ LIRIS, KU Leuven, Warmoesberg 26, 1000 Brussels, Belgium
{[estefania.serralasensio](mailto:estefania.serralasensio@kuleuven.be),[yannis.bertrand](mailto:yannis.bertrand@kuleuven.be)}@kuleuven.be

² Fraunhofer Institute for Applied Information Technology FIT,
Sankt Augustin, Germany
daniel.schuster@fit.fraunhofer.de

Abstract. Smart spaces' systems help users in their daily routines by automating various tasks. It can however be frustrating for users if the system does not evolve to support changes in their routines. To address this issue, we present an approach that combines two state-of-the-art approaches: *MatE*, an end-user model-driven approach, and *Cortado*, an incremental process mining approach. *CortadoMatE* can automatically detect changes in routines and allows the users to easily include the changes in the system step by step, or to even adapt them further before integrating them. In this way, continuous system evolution is addressed enabling the system to stay up to date with user needs.

Keywords: Process mining · Incremental process discovery · Task models · Smart spaces · Routine evolution

1 Introduction

Software evolution is necessary to ensure that software systems stay up to date in supporting changing user needs. This is essential for smart spaces, where the system continuously interacts with the home inhabitants.

We can mainly find two strategies for supporting system evolution in the literature: 1) end-user approaches, also called user-in-the-loop, provide easy-to-use interfaces for the end-users to describe the necessary changes in the system; while 2) user behaviour monitoring combined with data mining techniques can automatically identify changes in user behaviour (via, e.g., process checking), changes that can then be used to make the system evolve.

End-user approaches have the advantages that they allow the user to take part in the design of the system; hence, the system typically fits them more closely. Also, the users decide themselves when a change is needed. However, it may be tedious for the users to have to identify and describe these changes

Supported by KU Leuven internal funding.

© Springer Nature Switzerland AG 2022

A. Marrella and B. Weber (Eds.): BPM 2021 Workshops, LNBIP 436, pp. 391–402, 2022.

https://doi.org/10.1007/978-3-030-94343-1_30

themselves [13]. Monitoring methods, on the other hand, have the advantages that they can automatically detect changes and provide them in a formalism ready to be used to update the system. However, they usually need quite some historical data on previous executed tasks to be able to identify changes, which may sometimes do not really fit the user needs.

In [11,13], a user-in-the-loop approach, called MAtE, was presented to address runtime requirements evolution. MAtE uses state-of-the-art executable task models to support 1) the automation of user routines (i.e., set of tasks that are routinely executed in the same way, also known as behaviour patterns) and 2) the evolution of the automated routines by allowing users to describe the necessary changes using an end-user tool. Task models are used to describe routines at a high level of abstraction. At runtime, these models are interpreted to execute the routines the users desire to have automated by their smart homes. At any time during the execution of the system, the user can change the task models and, thus, the system's behavior, using the end-user tool.

In this paper, we propose to combine MAtE with Cortado [9], a state-of-the-art incremental process discovery approach that automatically detects newly observed behavior and allows to incrementally incorporate it. Using Cortado-MAtE, the user has the option to step by step decide which new behaviour should be incorporated into a task model. Once the changes are approved by the user, the task models are modified accordingly to support the approved changes.

The rest of the paper is organized as follows. Section 2 presents MAtE and Cortado. Section 3 describes how these approaches are integrated in order to improve the support for user routine evolution. Section 4 presents a case study of CortadoMAtE. Section 5 describes the related work and finally Sect. 6 concludes the paper and introduces some lines for future work.

2 Background

This section describes MAtE and Cortado, the two approaches were this paper is based on, as well as the corresponding modelling formalisms that the use.

2.1 MAtE: Executable Task Models

Executable task models [11,13] specify how a smart environment system can support the routines of its users. Task models are based on Hierarchical Task Analysis (HTA) [15], which are tree structures that refine high-level tasks into executable ones. Using executable task models, every user routine is described as a hierarchy of tasks. The root task represents the routine as a whole and is broken down into subtasks, which can be composite (they are further broken down) or executable (they are leaf tasks). Two types of refinements can be used to break down a composite or root task: exclusive refinement and temporal refinement. Exclusive refinement (represented by a solid line) decomposes a task into a set of subtasks in such a way that exactly one subtask will be executed; while using a Temporal refinement (represented by a dashed line) all the subtasks shall be

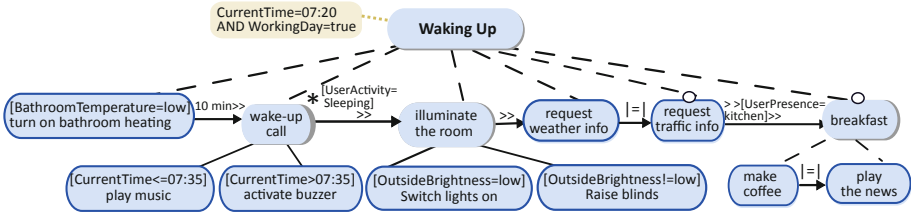


Fig. 1. Example task model for a wake up routine

performed following the order that is graphically depicted by the arrows between sibling tasks. Temporal constraints use Concurrent Task Trees (CTT) operators [8]. For example, in Fig. 1, we use:

- (T1 \gg T2): task T2 is triggered when task T1 finishes.
- Task Independence (T1 | = | T2): T1 and T2 can be performed in any order.

The task refinement process terminates when every leaf task in the tree is associated with a pervasive service (controlled by the smart environment system), which is capable of executing the task. The execution of a routine is context adaptive. Context can be described in a task model in the following constructs:

- Activation condition: it is associated with the root task of each routine, and indicates when the routine gets activated.
- Task precondition: it can be associated with a task to indicate that its execution depends on whether a situation holds. This condition is not expressed graphically in order to keep the diagrams easy to read.
- Iterative task: it is executed repeatedly while the situation associated with the task holds. Iterative tasks are graphically marked with an asterisk.
- Temporal constraints:
 - T1 \gg [s] \gg T2: after the completion of T1, T2 is started as soon as situation *s* holds.
 - T1 *t* \gg T2: after the completion of T1, T2 is started as soon as the time period *t* has elapsed.

Once the routines are specified by a system analyst, they can directly be executed by MAtE. MAtE is an automation engine that builds on the services provided for the smart environment to control the needed devices. We consider a service as a mechanism that provides a coherent set of functionalities described in terms of atomic operations (or methods). These operations allow the system to control the devices of the environment in order to change it and/or sense it. MAtE uses these services to perform the tasks of the routines specified in the task models and to sense context changes. MAtE monitors the context and executes the routines specified in the task model when their corresponding activation condition holds. The execution of the routines is performed in a context-adaptive way according to their specification and the current context. An end-user tool

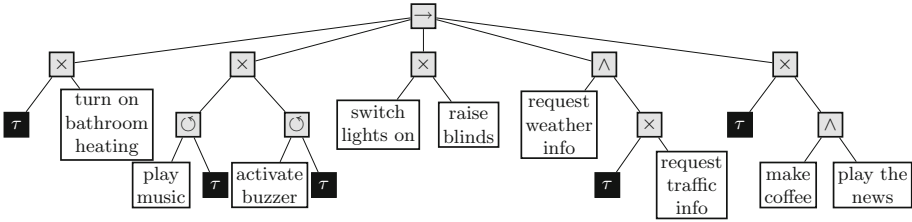


Fig. 2. Process tree modeling the control flow of the task model shown in Fig. 1

is also provided by MAtE to allow end-users to manually evolve the automated routines as they please using the available services [13]. Using this tool, any change that respects the task metamodel syntaxes can be performed to evolve the supported routines. Thus, new behaviour can be created, or those that are already specified can be modified or deleted. Finally, users can also execute the smart environment services using a web User Interface (UI) [13].

2.2 Process Trees and Incremental Process Tree Discovery

Process models allow us to model the control-flow of activities performed within the execution of a process. Process trees [9] are an important process model formalism within the area of *process mining* [1]. Figure 2 shows an example of process tree that represents the control-flow of the task model shown in Fig. 1. Leaf nodes of a process tree represent activities, which can either be visible, e.g., “turn on bathroom heating”; or invisible, τ leaf nodes used to model certain control-flow patterns. Inner nodes of a tree represent operators that specify the control-flow relation among its subtrees. We distinguish four different operators:

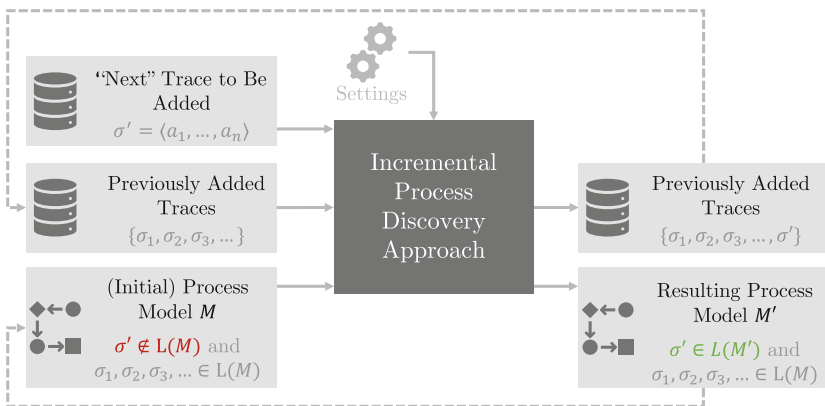


Fig. 3. Overview of the incremental process discovery algorithm [9]. Figure from [10]

1. Sequence (\rightarrow): subtrees must be executed in the given order
2. Exclusive-choice (\times): exactly one subtree must be executed
3. Loop (\odot): always consists of two subtrees. The first subtree (do-part) must be executed, afterwards the second subtree (redo-part) is optionally executed. If the second subtree is executed, the first subtree must be executed again.
4. Parallel (\wedge): subtrees can be executed in any order and also interleaved

Compared to task models, as most process modelling formalisms used in process mining, process trees purely focus on the control-flow of activities, i.e., no contextual information is present in process trees. In Fig. 1, for instance, the task “turn on bathroom heating” is only executed if the temperature is low. However, in the corresponding process tree (Fig. 2), the activity “turn on bathroom heating” is modelled as an optional activity, i.e., either “turn on bathroom heating” or τ is executed.

Process discovery is an important sub-discipline in process mining [1] and covers techniques to learn a process model from observed process behaviour. Recently, an incremental process tree discovery approach, has been introduced [9] and implemented in a software tool [10] called *Cortado*. A conceptual overview of the approach is given in Fig. 3. Starting from an initial process tree M , Cortado allows a user to incrementally extend/learn a process tree from observed process behaviour. Cortado identifies, in an event log, the executed traces that are not yet represented in the initial process tree. These traces are then shown ordered by frequency to the user, who can select the traces that s/he wants the process tree to also support. By incrementally selecting a trace σ' , which represents a single execution of the process, the approach modifies the tree M into M' such that the selected trace and the previously added traces are accepted by the

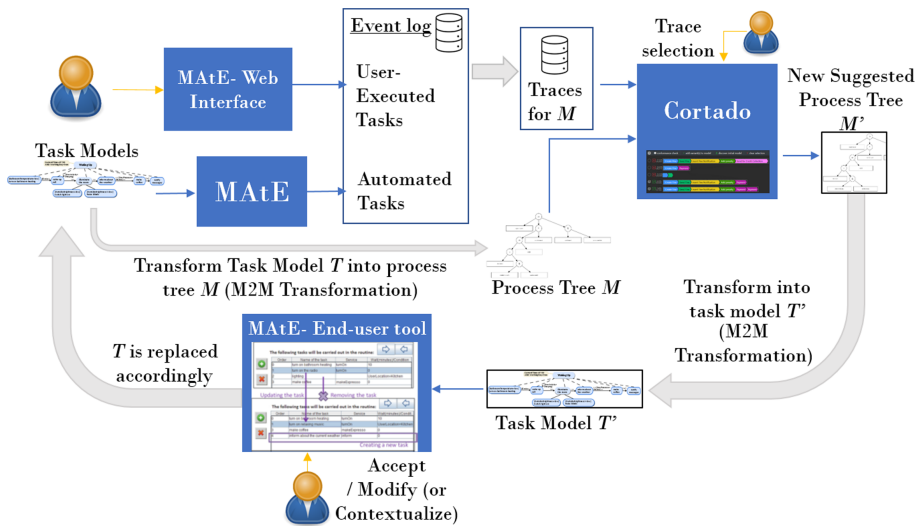


Fig. 4. Approach for continuous evolution of user automated routines

altered process tree, i.e., visualised by $\sigma' \in L(M')$ and $\sigma_1, \sigma_2, \sigma_3, \dots \in L(M')$ in Fig. 3. The tree M' is then used as an input in the next incremental execution.

In short, the incremental approach allows a user to gradually discover/learn a process model from observed process behaviour. Moreover, a user can extend an existing process model any time to represent new observed behaviour.

3 CortadoMAtE: Addressing Evolution

To support end-users keeping the automation of their routines updated according to their needs, we integrate MAtE with Cortado (see Sect. 2). Figure 4 shows the overall approach of CortadoMAtE. As explained, Cortado needs as input: a) an (initial) process tree, and b) the new traces that are not yet supported by the process tree (*Traces for M* in Fig. 4). In order to provide the necessary inputs, MAtE and the web interface are extended to record the execution of routines (i.e., traces) in the required format. The tasks executed for a routine by MAtE are recorded with an associated *case ID*, which identifies the routine execution to which the executed task belongs. The web interface additionally records the tasks that the user executes via the web interface; these tasks do not have a predefined case ID as they do not belong to any routine yet.

In addition, we need to transform the initial task models into process trees. For this purpose, we define model to model (M2M) mappings as shown in Fig. 5.

	TASK MODEL	PROCESS TREE
Task		
Directly follows		
Parallel		
Optionality		
Exclusive choice		
Loop		

Fig. 5. Main mappings between task models and process trees. Note that the dotted tasks in the table can represent either a leaf or a composite task; also, M2M represents the recursive application of mappings for that particular task.

E.g., in the second and third row of the table, we see that the temporal constraints \gg and $| = |$ in task models are mapped to \rightarrow and \wedge respectively. As can be seen, composite tasks (of task models) are directly translated into operators in the process tree, i.e., the translation of a composite task corresponds to the translation of its subtasks. As such, mappings are applied in a recursive manner. Also note that context information is omitted in the mapping, as Cortado does not yet support it (see Sect. 6). To enable the reverse mapping (from process tree to task model), context information as well as names of composite tasks are stored as labels in the corresponding process tree tasks or operators.

Taking into account these two inputs, CortadoMAte detects changes on the current routines using the incremental process tree discovery algorithm. For each task model, we first extract the traces that belong to the execution of that model. Since usually routines are time-bounded, we consider that tasks without case ID belong to the trace that is closer in time. As explained in Sect. 2.2, the identified changes are shown to the user in the form of traces. The user can then select the traces that should also be supported. Once this is done, a new process tree (*New Suggested Process Tree M'* in Fig. 4) is obtained.

This process tree is then transformed back into a task model using the M2M mappings. The user can then select to 1) accept the new routine or 2) modify the suggested routine using the user-interface, which also allows the user to contextualize it if s/he so prefers. The final task model will then replace the old one. Since task models are interpreted at runtime, the system reflects the changes from the next execution of the routine [11]. In short, CortadoMAte is a semi-automated evolution approach in which the user receives suggestions for changes, but still has the control over the evolution supported by the underlying incremental process discovery approach.

4 Case Study

This section presents a case study of the proposed approach. First, we introduce the experimental setup. Subsequently, we discuss the results.

4.1 Experimental Setup

Figure 6 shows an overview of the experimental setup. The overall idea is to demonstrate that our approach is able to incrementally discover a task model that represents the new observed behaviour. Therefore, we start from a target

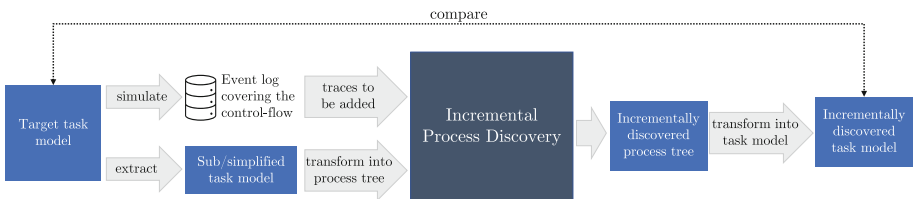


Fig. 6. Overview of the experimental setup

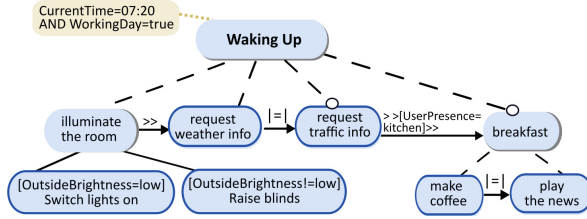
Table 1. Excerpt of the simulated event data from the target task model (Fig. 2). Table shows one execution of the waking up routine (i.e., a trace), and the start of a next execution of the same routine.

<i>Control-flow information</i>			<i>Context information</i>				
Case-ID	Activity/Task	Timestamp	Bathroom temperature	User presence	Outside brightness	User activity	Actor
100	turn on bathroom heating	01-01-2000 07:20	low	bedroom	low	awake	system
100	illuminate the room	01-01-2000 07:30	medium	bedroom	low	awake	system
100	request weather info	01-01-2000 07:31	medium	bedroom	low	awake	system
100	-	01-01-2000 07:33	medium	bathroom	low	awake	user 1
100	make coffee	01-01-2000 07:52	medium	kitchen	low	awake	system
100	play the news	01-01-2000 07:52	medium	kitchen	low	awake	system
101	turn on bathroom heating	01-02-2000 07:20	low	bedroom	low	sleeping	system
...

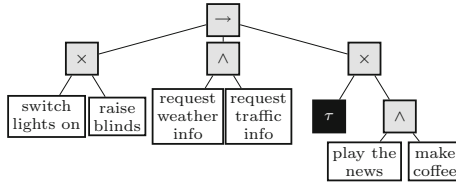
task model that we would like to incrementally discover with our approach. First, the target task model is simulated to obtain event data capturing the different possible executions of the task model. Next, from the target task model, we extract a simplified task model and transform it into a process tree (see Sect. 3). This process tree serves as an initial model for the incremental process discovery approach. Now, we apply incremental process tree discovery. Thereby, we incrementally extend the simplified task model, which was transformed into a process tree, by new process behaviour from the simulated event log. Finally, we obtain an incrementally extended process tree. This process tree is converted back into a task model and compared to the target task model.

As a target task model we use the one introduced in Fig. 1. First, we simulate this target task model as stated in Fig. 6. Table 1 shows an excerpt of the event data generated. Each row represents an event. Events with the same case-ID belong to the same execution of the task model, e.g., events with case-id 100 represent the execution of the task model on 01-01-2000. Table 1 is divided into control-flow information and context information. Next, we extract the traces out of the event data, e.g., the case 100 describes the trace $\langle \textit{turn on bathroom heating, illuminate the room, request weather information, make coffee, play the news} \rangle$. These traces are then incrementally fed into the incremental process tree discovery approach. In total, we simulated the target task model 1,000 times, i.e., we obtained 1,000 traces and a total amount of 7,028 events. Moreover, the simulated event data contains 345 different trace-variants. Note that multiple traces can describe the same sequence of executed tasks.

From the target task model (Fig. 1), we extract a simplified task model which serves as an initial model. Figure 8a shows the simplified task model, and Fig. 7b shows the corresponding process tree representation. Note that, for instance, the activity *illuminate the room* that we observe in the generated event data (Table 1) is not part of the initial task model (Fig. 8a).



(a) Initial/simplified task model



(b) Initial/simplified process tree

Fig. 7. Initial/simplified task model extracted from the target task model (Fig. 1) and corresponding process tree representation

4.2 Results

This section presents the results of the conducted experiment. We show the different obtained process trees and added traces after each incremental execution of the incremental discovery approach. Finally, we compare the resulting task model with the target task model as visualized in Fig. 6.

Figure 8 shows the intermediate trees during the incremental execution. Starting from the initial task model (Fig. 8a) that is converted into a process tree (Fig. 7b), we add the trace $\langle \text{switch light on}, \text{request weather info}, \text{play the news}, \text{make coffee} \rangle$, consider Fig. 8a, which is obtained from the simulated event data, as partly shown in Table 1. After adding six traces to our initial model, we obtain the process tree shown in Fig. 8d. This process tree describes the entire event data that we obtained through simulation of the target task model. The experimental results show that it is possible to extend an initially given task model by new behavior such that the resulting model represents the intended behavior.

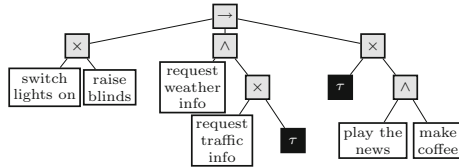
5 State-of-the-Art in Smart Home System Evolution

There exist three types of approaches to smart space systems' evolution: User-in-the-loop, monitoring with data driven, and combined techniques.

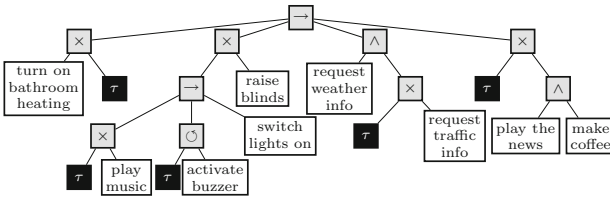
User-in-the-loop Approaches. User-in-the-loop approaches provide easy-to-use interfaces to allow less technically-fluent users to modify their habits [6]. For instance, Kolb et al. propose techniques for end-users to define and modify process models using CTTs [5]. Similarly to MAte, Koussaifi et al. [6] developed a framework and tool based on model-driven engineering (MDE) to allow users

to modify their smart home systems. They can accept or reject models of applications, automatically generated for accepted application proposals. The main drawback of these approaches is that the users need to do the evolution process themselves: they need to identify the changes and they need to introduce these changes in the system using the provided tools.

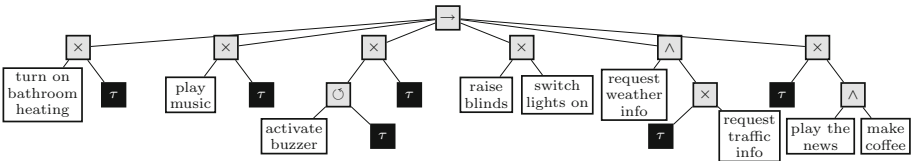
Monitoring and Data-Driven. Monitoring and data-driven approaches use data generated by the use of the system to detect changes in user routines. Most monitoring-based models follow Machine-learning approaches, based on



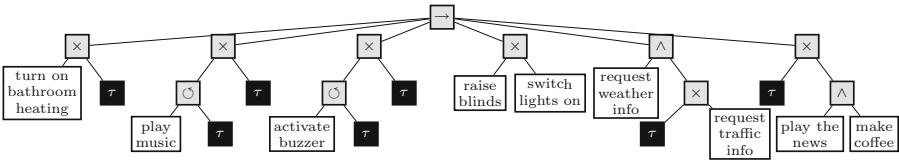
(a) Added trace \langle switch light on, request weather info, play the news, make coffee \rangle



(b) Added traces: \langle turn on bathroom heating, raise blinds, request weather info, make coffee, play the news \rangle and \langle turn on bathroom heating, play music, activate buzzer, activate buzzer, switch light on, request weather info, request traffic info, play the news, make coffee \rangle



(c) Added trace \langle turn on bathroom heating, play music, activate buzzer, activate buzzer, switch light on, request weather info, request traffic info, play the news, make coffee \rangle



(d) Added trace \langle play music, play music, play music, raise blinds, request weather info, request traffic info, play the news, make coffee \rangle

Fig. 8. Incrementally discovered trees starting from the initial one (Fig. 7b)

mathematical and statistical formalisms to model the routines by detecting patterns in the collected data. These approaches have been extensively analysed by Leotta et al. [7]. E.g., MavHome and CASAS [3] use Hidden Markov Models to represent user behaviour as patterns of series of states linked by transitions with given probabilities. The models can be extended with new states to support new user behaviour. Dabrowski et al. [4] use process discovery techniques to detect deviations from the designed requirements. Main drawbacks of these techniques include: 1) they require a great amount of training data (the cold-start problem), hence it may take some time for the system to recognise a habit change, delaying the support of the user's habit by the system [14]; 2) The absence of knowledge on users' actual desires may lead to automating tasks that the user does not wish to automate [13]; and 3) Only tasks that have already been performed, and only in the way they have already been performed, can be learned [13].

Combined Approaches. Some authors propose to combine monitoring approaches with so-called knowledge-driven approaches [2, 16]. In these papers, a knowledge base is first constituted to broadly describe the routines performed by the user. A data-driven model is then learned, which better fits the actual routines of the user. The resulting model is often better at recognizing activities [2] Sukor et al. [16] notes that the “cold start” effect is mitigated. However, these techniques are designed for activity recognition, not for smart home evolution.

6 Conclusion and Future Work

In this paper, we have presented CortadoMAtE, an approach that does not only support the automation of routines in smart spaces, but also facilitates the continuous evolution of the automated routines. CortadoMAtE tackles the problems of previous approaches, such as the cold-start problem. Moreover, although CortadoMAtE suggests possible changes to the users, they always have control over the routines that are finally automated; therefore, both users' desires as well as time and energy concerns are taken into account [13]. Task models can be stored, so users can easily revert to previous versions of the system if they are not satisfied with a newly implemented automation. In addition, using the end-user interface provided by MAtE, it is possible to automate tasks regardless of whether or not the users have performed them in the past, which is a big advantage of CortadoMAtE over classical monitoring approaches. Finally, CortadoMAtE also requires less input from the user than typical user-in-the-loop approaches, making the CortadoMAtE more user-friendly.

As future works, the automated discovery of the context of the model should be addressed. Currently, the user manually contextualizes the models generated by Cortado, but the incremental process mining algorithm does not integrate context information yet. However, automating this step would 1) contribute to further reducing the input required from the user and 2) potentially extract unknown context patterns from the data. Context data are already collected by the sensors present in the smart home and could be utilized. In addition, when users are proposed a new task model, it could help them to be able to simulate its

execution. This would help in their decision to accept or reject the new model, and would also help them visualize the impact of manual modifications and contextualization. This simulation could be supported by using CPN Tools [12].

References

1. van der Aalst, W.M.P.: *Process Mining - Data Science in Action*. Springer, Heidelberg (2016). <https://doi.org/10.1007/978-3-662-49851-4>
2. Azkune, G., Almeida, A., López-de Ipiña, D., Chen, L.: Extending knowledge-driven activity models through data-driven learning techniques. *Expert Syst. Appl.* **42**(6), 3115–3128 (2015)
3. Cook, D., et al.: MavHome: an agent-based smart home. In: *PerCom 2003*, pp. 521–524. IEEE Computer Society (2003)
4. Dabrowski, J., Kifetew, F.M., Munante, D., Letier, E., Siena, A., Susi, A.: Discovering requirements through goal-driven process mining. In: *REW*, pp. 199–203. IEEE (2017)
5. Kolb, J., Reichert, M., Weber, B.: Using concurrent task trees for stakeholder-centered modeling and visualization of business processes. In: Oppl, S., Fleischmann, A. (eds.) *S-BPM ONE 2012*. CCIS, vol. 284, pp. 237–251. Springer, Heidelberg (2012). <https://doi.org/10.1007/978-3-642-29294-1-19>
6. Koussaifi, M., Arcangeli, J.P., Trouilhet, S., Bruel, J.-M.: Putting the end-user in the loop in smart ambient systems: an approach based on model-driven engineering. Technical report (2020)
7. Leotta, F., Mecella, M., Sora, D., Catarci, T.: Surveying human habit modeling and mining techniques in smart spaces. *Future Internet* **11**(1), 23 (2019)
8. Paternó, F.: *ConcurTaskTrees: An Engineered Notation for Task Models*. Lawrence Erlbaum, Mahwah (2004)
9. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Incremental discovery of hierarchical process models. In: Dalpiaz, F., Zdravkovic, J., Loucopoulos, P. (eds.) *RCIS 2020*. LNBP, vol. 385, pp. 417–433. Springer, Cham (2020). <https://doi.org/10.1007/978-3-030-50316-1-25>
10. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Cortado—an interactive tool for data-driven process discovery and modeling. In: Buchs, D., Carmona, J. (eds.) *PETRI NETS 2021*. LNCS, vol. 12734, pp. 465–475. Springer, Cham (2021). <https://doi.org/10.1007/978-3-030-76983-3-23>
11. Serral, E., Valderas, P., Pelechano, V.: Context-adaptive coordination of pervasive services by interpreting models during runtime. *Comput. J.* **56**(1), 87–114 (2013)
12. Serral, E., De Smedt, J., Snoeck, M., Vanthienen, J.: Context-adaptive petri nets: supporting adaptation for the execution context. *Expert Syst. Appl.* **42**(23), 9307–9317 (2015)
13. Serral, E., Valderas, P., Pelechano, V.: Addressing the evolution of automated user behaviour patterns by runtime model interpretation. *Softw. Syst. Model.* **14**(4), 1387–1420 (2013). <https://doi.org/10.1007/s10270-013-0371-3>
14. Serral, E., Valderas, P., Pelechano, V.: Improving the cold-start problem in user task automation by using models at runtime. In: Pokorny, J., et al. (eds.) *Information Systems Development*, pp. 671–683. Springer, New York (2011). <https://doi.org/10.1007/978-1-4419-9790-6-54>
15. Shepherd, A.: *Hierarchical Task Analysis*. Taylor & Francis, Milton Park (2001)
16. Sukor, A.S.A., Zakaria, A., Rahim, N.A., Kamarudin, L.M., Setchi, R., Nishizaki, H.: A hybrid approach of knowledge-driven and data-driven reasoning for activity recognition in smart homes. *J. Intell. Fuzzy Syst.* **36**(5), 4177–4188 (2019)