Real-Time Task Recognition for Mechanical Maintenance Procedures*

Marcus Behrendt[1] and Jan-Patrick Osterloh[1][0000−0002−4200−3002]

[1] OFFIS – Institute for Information Technology, Escherweg 2, 26121 Oldenburg, Germany
{behrendt, osterloh}@offis.de
http://www.offis.de

Abstract. Technical systems, like assembly lines or wind turbines, are usually set up at locations far away from their manufacturer. Because well-trained technicians for maintenance may not always be available at these distributed locations, we propose a solution that allows local technicians without detailed knowledge of the system to do the maintenance on their own, by the help of an assistance system (AS). The core component of the AS we focus on is the so-called task inference system (TIS) that recognizes the current task that the technician is doing within the maintenance procedure. The possible processing sequences during task processing are modeled as hierarchical task trees using the DCoS modeling language, which is based on ConcurTaskTrees [19].
Based on the recognition of the current task and processing step, the AS can provide information, indicate errors and provide help. TIS infers tasks in real time by using different kinds of sensors (eye-tracker, hand and gesture sensors, RFID for tool recognition, etc.) that observe the technician while performing the task. In this paper, we will show how we extended our task models, such that the task inference models based, which are based on the Dempster-Shafer theory [7], can be derived from these task models.

Keywords: Maintenance Task · Task Modeling · Task Recognition · Dempster-Shafer

1 Introduction

Maintenance is a success factor for all kind of companies. According to studies direct and indirect cost of maintenance amount to 40% [18], but done right it has the potential to avoid follow-up costs five times higher that would arise through machine fault [1]. Just by improper maintenance, additional costs of 14 billion Euro occur annually in Germany [18].

* The research leading to these results was funded by the Lower Saxony Ministry of Economics, Labour, Transport and Digitisation and the Lower Saxony Ministry of Science and Culture as part of the IKIMUNI initiative as well as funded as part of the Cocomo project by the NBank and the EU EFRE Program.

In a typical factory building a lot of technical installations like assembly lines, punching machines, printers, etc. have to be maintained both scheduled and unscheduled. A share of them are bought and often there is no local technician available that has the required knowledge of the installation to be able to perform proper maintenance. As a result, companies spend an average of 35% of their maintenance costs on third-party personnel [3].

But relying on third-party personnel is prone to be critical: It is often not available in a certain amount of time. And if the maintenance task is a repair this would inevitably cause high costs by a protracted loss of production. In such a situation companies have the choice to either wait for the third-party personnel to be available or try to get the repair done by a local technician that is not familiar with the installation. The latter has the potential to aggravate the situation even more and can be dangerous for man and machine caused by the insufficient knowledge of the installation, as well as potentially breaking warranty.

But not only maintenance of technical installations inside factory buildings can be an issue. Technical installations that are scattered over vast areas, like wind turbines, transmission towers or fuse boxes in households, have to be maintained, too. Usually a single technician or a small team is sent to them to do the maintenance. In most cases the technicians should be experienced enough to properly maintain such installations. However, the technical complexity in such installations is constantly increasing and that carries the risk that technicians are less familiar with certain rare maintenance routines of the installation. Again, this situation would cause high costs through a loss of production if the maintenance is a repair that cannot be performed by the technicians on site because of missing knowledge. In such a situation the only options are to either read documentary or to call for help from other technicians – both cost time.

In this paper, we show a solution that enables companies to reduce costs through losses of production by allowing local technicians to perform maintenance procedures for technical installations of the same quality as well trained third-party personnel. In addition, our solution might also be suitable for the training of technicians.

Roughly speaking our solution is to create an assistance system (AS) that supports technicians with performing maintenance procedures. In our work we primarily focus on the core component of the AS – the task inference system (TIS). The TIS keeps track of that what the technician is doing. To realize such a component we combine two different approaches from two different research areas into one integrated approach.

On the one hand, we make use of hierarchical task models. Hierarchical task models are used to analyze and describe complex tasks by decomposing them into smaller and smaller units down to a certain level of detail [8]. There are several formal languages for hierarchical task models. We chose to use the DCoS-XML description language [19] because it allows us to model temporal constraints with a high level of detail. On the other hand, we build upon task recognition models from Honecker & Schulte [13,12,22,15]. They used the Dempster-Schafer

theory (DST) to develop task recognition models for highly parallel tasks with little temporal dependencies. We extend this approach by integrating it with hierarchical task models to improve task recognition for tasks with high temporal constraints, which is typical for maintenance tasks.

Based on the TIS the AS is able to give situation-aware information to the technician like highlighting important parts on an Augmented Reality device. The AS can also warn the technician about skipped tasks. The most important part is that the AS provides help if the technician does not know further. The help could be as insight into useful documents or connecting with an expert. The advantage here is that the technician does not need to explain what he is doing – the AS would find suitable documents and would connect to an appropriate expert based on the Information the TIS provides.

In addition to the real use at work, it is also conceivable to use such an AS for the training and education of technicians. This has a great learning effect through learning-by-doing as the system offers information, alerts and help to the trainee. Incidentally, instructors would be able to teach larger groups of trainees efficiently.

The structure of the paper is as follows: In section 2 we provide a brief overview of related work on hierarchical task models and task recognition. Our concept of combining the two approaches is presented in section 3. In section 4 we demonstrate our combined approach with a use case example. Section 5 includes a discussion of the achieved results.

2 Related Work

There are several approaches to formally describe tasks and procedures. An easy to use approach is Hierarchical Task Analysis (HTA) [23]. It is easy because no special tooling is required - HTA can be done by pen and paper. It allows to decompose tasks into subtasks revealing their order and hierarchical structure.

Another, early widespread approach is the GOMS (Goals, Operators, Methods, Selection) description language [4]. It allows to define user models for specific tasks and has often been used to calculate execution times for tasks.

ConCurTaskTrees (CTTs) [20] are a more recent approach. CTTs are a more general model and focus not only on a single user, but are able to describe interactions between multiple users and multiple system agents. In contrast to HTA, temporal relationships can be described in more detail with CTTs, for example, sequential, parallel, or concurrent processing of work steps. CTTs are quite expressive and have been adopted by several modeling tools and languages like CTTE (ConcurTaskTrees Environment), HAMSTERS [11] or the DCoS-XML (dynamic, distributed, cooperative systems) [19].

Task recognition in the literature often appears under the term “activity recognition” [5,16,17,24]. However, these are more related to living environments than to maintenance tasks in the technical domain. Here different kinds of Dynamic Bayesian Networks (DBN) are used intensively.

Donath studied task recognition for military helicopter missions [9,10]. In her scenario a helicopter crew is on a mission and has the ability to controls several unmanned aerial vehicle (UAV). The mission is decomposed with the HTA method [23] into single tasks like "control speed", "check flight height" or "make radio message". To recognize the single tasks Hidden Markov Models (HMM) [21] are used. Here the tasks represent hidden states and are connected to suitable observations. The observations are semantically enriched sensor information like "pilot looks on display X", "pilot uses microphone" or "pilot increases air speed". The models work quite well if tasks are done in a normative manner but tend to fail when tasks are done in a non-normative way [9].

Honecker and Schulte [12,13,14,15,22] successfully tried to circumvent this deficit by creating a simplified Dempster–Shafer theory for task recognition applicable in the same scenario. Each task in the decomposed mission is implemented as a single binary Dempster-Shafer model. Here a task is assigned two contrary hypotheses. The first hypothesis states that the task is currently done, whereas the second hypothesis states that the task is currently not done. A hypothesis is assigned evidences that support it. When inferring, a task has three values as output: The belief and the doubt that the task is currently done and the ignorance about whether the task is done or not.

Using DST for task recognition has a major advance to other probabilistic methods specially Bayesian Networks [6]: The number of model parameters is rather low and thus models can easily be proposed by experts in addition of applying machine learning and they are human readable. Additionally, the uncertainty that is a core element of DST can be very useful for an AS: A better appropriate strategy for an AS is to inform the human that it is not knowing what the human is doing instead of doing wrong things that come from wrong assumptions. This is going to frustrate humans and might further lead to dangerous situations.

But the work of Honecker also has one major drawback for task recognition in the technical domain: Tasks in helicopter missions tend to be strongly parallel. A pilot can check air speed before making radio messages or vice versa. The pilot can even do these both things at the same time. The order of the tasks does not play a role because aviation basically consists of monitoring tasks. But in the technical domain the tasks tend to be much more sequential. Imagine the scenario where a technician wants to exchange a broken fuse in a fuse box. When we want to recognize the task "insert new fuse" we should have recognized some tasks before that are preconditions: The technician first has to do the task "open fuse box" and "remove broken fuse" before he is even physically able to insert the new fuse. Honecker's approach does not consider these dependencies between tasks.

In the next section we will show how to apply Honecker's approach to strongly sequential tasks by combining CTT with DST.

3 Combining Methods

As mentioned before, Honecker developed a simplified DST. The simplification is that each task is represented in a separate DST model rather that all task are represented in a single DST model. Although the tasks are depicted in separate models, we will refer to the set of all tasks and their relationships as a composite model.

A composite model $M = (T, \Lambda, E)$ in Honecker's simplified DST consists of a set of n tasks $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$, a set of m events $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ and a set of evidences $E \subseteq T \times \Lambda$. The set of tasks T is the result of the decomposition of an overall task into single elementary tasks. The set of events Λ is defined on all possible data of sensors and data sources like eye trackers, microphones, speedometers, or acceleration sensors. An event may be something like "pilot looks on X", "pilot makes radio call", "aircraft's speed is low" or "aircraft's height is over 1000 m". Whether an event is currently active or not active is given by $a : \Lambda \to \{0, 1\}$. A task $\tau \in T$ may be connected with an event $\lambda \in \Lambda$ through an evidence $e_{\tau,\lambda} = (\tau, \lambda) \in E$. An evidence $e_{\tau,\lambda}$ either supports or doubts the hypothesis that task τ is being done when event λ is being active. This is given by $s : E \to \{0, 1\}$. How strong the support or the doubt is depends, as in the classical theory, on the mass of the evidence. The mass of an evidence e is given by $m : E \to [0, 1]$. The set of evidences assigned to a task τ is denoted as $E^\tau \subseteq E$. The components that build up a composite model M are depicted in Figure 1. Tasks T one the left side are connected to events Λ on the right side through evidences E in the middle. Each evidence e supports or doubts a task with $s(e)$ by a mass of $m(e)$.

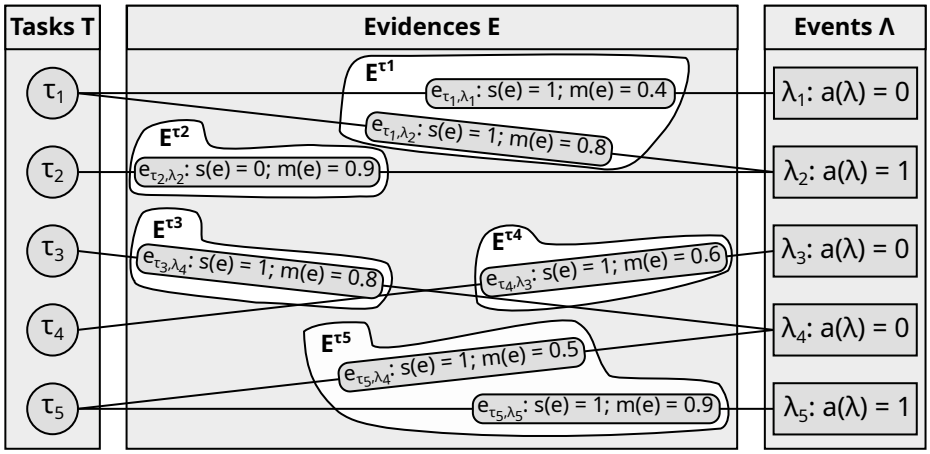


Fig. 1. Components of a simplified DST model M on an example. A task $\tau_i \in T$ on the left side and an event $\lambda_j \in \Lambda$ on the right sight are connected through an evidence $e_{\tau_i,\lambda_j} \in E$. Each evidence e supports or doubts a task with $s(e)$ by a mass of $m(e)$.

The execution time of inferring a model in the classical DST increases exponentially with the number of hypothesis as an evidence can be assigned a whole set of hypothesis [2]. This is the reason why the classical DST has been rarely used in applications. Honecker has simplified his research question to use DST for task recognition. The question is no longer "Which of all possible task combinations the pilot is currently working on?" but "Is the pilot currently working task X? Yes or no?" [13]. So in Honecker's DST an evidence can only be assigned one out of two possible hypotheses. This is either the hypothesis that states that the task is being done or the hypothesis that states that the task is not being done. Honecker refers to his kind of simplified DST as a binary DST. The simplification makes DST suitable for real time applications because less computations are needed when inferring [13].

Inference is the process of iterating through the evidences $\{e_{\tau,\lambda} \in E^{\tau} | a(\lambda) = 1\}$ of a task τ that have active events and combining the masses of these evidences to a belief triplet $\vartheta_{\tau} = (p_{\tau}, q_{\tau}, r_{\tau}) : p_{\tau} + q_{\tau} + r_{\tau} = 1$ by the application of Dempster's Rule of Combination [7]. The value p_{τ} indicates the belief that the task τ is currently done. The value q_{τ} indicates the doubt that the task τ is currently done. The ignorance r_{τ} indicates uncertainty about whether the task τ is done or not. Inference is performed in a fixed time interval for each task $\tau \in T$.

This is the basic concept of Honecker's simplified DST. But as explained in the previous section Honecker's approach is not well suitable for sequential maintenance tasks but rather for strongly parallelized tasks like helicopter missions. To circumvent this deficit we extend DCoS-XML models and use them in conjunction with Honecker's approach.

DCoS-XML enables us to hierarchically model tasks and to encode all possible sequences of a task. The hierarchical structure of a task is represented in a tree. Each task can be divided into subtasks. A subtask itself is also a task and in turn can have subtasks. The hierarchic structure can be seen in Figure 2. Here task τ_2 and task τ_3 are subtasks of task $\tau_{2,3}$. Task $\tau_{2,3}$ is itself a subtask of task $\tau_{1,2,3,4,5}$ (together with τ_1, τ_2 and τ_4).

To encode sequences two sibling tasks τ_i and τ_j are connected with an edge containing a temporal operator. A temporal operator defines in which order the two tasks must or can be executed. There are several different temporal operators like *enabling*, *parallel* or *optional* described in [20]. Temporal operators are illustrated in Figure 2. The tasks τ_1 and $\tau_{2,3}$ are connected through an edge with an *enabling* operator. It states that task $\tau_{2,3}$ cannot be started before task τ_1 has been finished. The edge from task τ_2 to task τ_3 contains an *interleaving* operator. This means that the execution can switch between these tasks permanently. At last there is an *independence* operator between the tasks τ_4 and τ_5. This operator states that one of the two tasks has to be started and finished before the other task can be started.

We use DCoS-XML to extend Honecker's DST approach in order to improve task recognition for tasks with more complex temporal constraints like maintenance tasks. It works well because the decomposed tasks in the DST model and

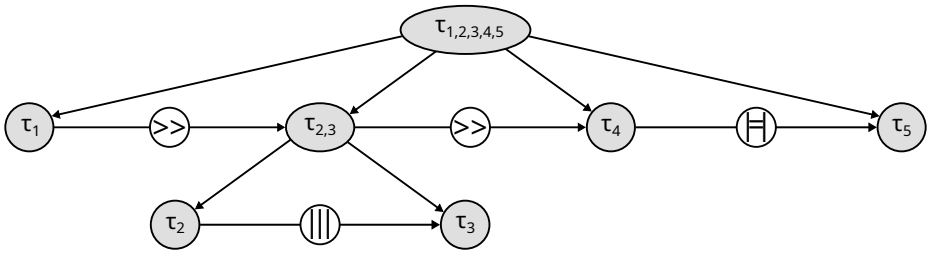


Fig. 2. Example DCoS-XML model depicting the hierarchical tree structure of a task and the temporal constraints between tasks. There are three kinds of temporal operators in the figure: An *enabling* operators between the tasks τ_1 and $\tau_{2,3}$ and the tasks $\tau_{2,3}$ and τ, an *interleaving* operator between the tasks τ_2 and τ_3, and an *independence* operator between the tasks τ_4 and τ_5.

the leaf tasks in the DCoS-XML model depict the same facts - both describe elementary tasks that can not be further decomposed. So tasks can be mapped from a DST model into a DCoS-XML model as leaf tasks. The mapping allows them to be recognized in the DCoS-XML model using the DST inference algorithm. It is sufficient to only recognize leaf tasks in the DCoS-XML model. By recognizing a leaf task, in turn, the parent task is recognized. In addition, the activities associated with events are defined very demarcated in leave tasks compared to parent tasks. The combined approach is roughly:

1. Propagate results inferred from DST model to a rule engine operating on the hierarchal DCoS-XML model that ...
2. ... keeps track of tasks' states by considering both sequential constraints and results inferred from the DST model and ...
3. ... that in turn is able to modify DST model parameters during the term to improve the inferred results from the DST model.

The combined approach is visualized in Figure 3. From each belief triplet ϑ_τ we first generate a belief value that is $\beta_\tau = p_\tau - q_\tau$. We use these belief values to define an order relation on the set of tasks T. We will describe how the rule engine recognizes the currently performed task τ_t by evaluating the order of T later in this section.

To decide which task is currently performed, the rule engine alters the states of the tasks within the DCoS-XML model. A task accepts the following states: *waiting*, *active*, *blocked*, *ongoing*, *done*. A state machine depicting the states of a task τ and the state transitions are given in Figure 4. A task is *blocked* if it waits for another task to be *done*. A task is *waiting* if it is not blocked by another task. A task is *active* when it is currently executed. Parent tasks are *ongoing* when a subtask is *active*. A task is *done* when its execution is finished or all of its subtasks are *done*.

To set the tasks' states the rule engine uses the belief values $(\beta_{\tau_1}, \beta_{\tau_2}, \ldots, \beta_{\tau_n})$ from DST inferring. The rule engine picks the highest belief value β_τ amongst the

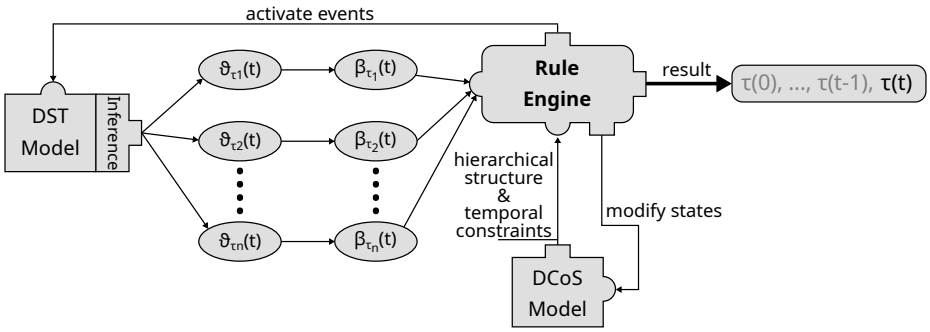


Fig. 3. Visualization of the recognition of a task τ at a time t with our combined approach: At first the DST Inference create belief triplets $(\vartheta_{\tau_1}, \vartheta_{\tau_2}, \ldots, \vartheta_{\tau_n})$. The belief triplets are then transformed into belief values $(\beta\tau_1, \beta\tau_2, \ldots, \beta\tau_n)$. The Rule Engine uses the highest value amongst the belief values and sets the appropriate task as the current one $\tau(t)$ and thus modifies the states of the tasks in the DCoS-XML model. At last the Rule Engine activates the doubting evidence for task $\tau(t)$

belief values to set the appropriate task τ to *active*. After setting a task to *active* the previous *active* task is set to *done* or *ongoing*. In this way the rule engine is able to track the sequence of tasks $(\tau(0), \ldots, \tau(t-1), \tau(t))$ that a technician is producing while performing a maintenance task. Based on the knowledge of the temporal constraints the rule engine is able to recognize whether a technician has skipped a task and can produce a warning in that case. The warning could be used by the overlying AS, e.g. ask the user if the skip was intended. This can be illustrated by an example in Figure 2. The edge from task τ_1 to task $\tau_{2,3}$ contains an *enabling* operator. If the DST inference in the first run would assign task τ_2 (subtask of $\tau_{2,3}$) the highest belief value then the rule engine would set task τ_2 from *blocked* to *ongoing*. But task τ_1, that compellingly should be completed before τ_2, is still *blocked*. Such situations can always occur for a variety of reasons, such as sensor errors or carelessness on the part of the technician if he forgets to do a task.

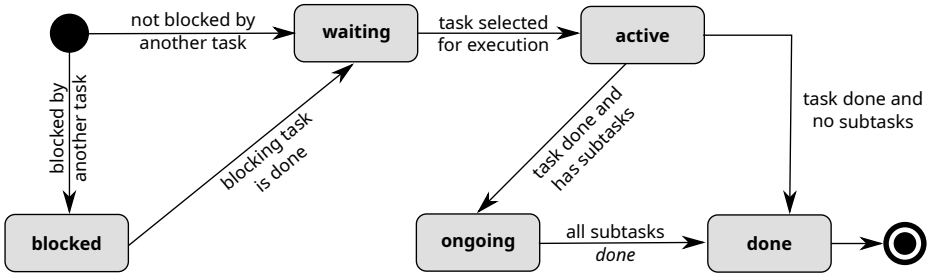


Fig. 4. The modifications the Rule Engine makes to a task illustrated by a state machine

At last there is the problem of recognizing tasks again that are already set to *done*. This could happen because of sensor error and is seen as a false recognition. But in some cases doing a task again after it has finished is intentional. Imagine a technician that has finished task τ_2 in Figure 2 and now is currently working on task τ_4. He is suddenly realizing that he forgot to do some parts that belong to task τ_2. So he has to stop working on task τ_4 and do the forgotten parts of task τ_2. In future work we want to recognize that task τ_2 is *active* again. So we cannot just blacklist all *done* tasks. Instead we want to reduce the probability of recognizing a task again after it has been done. This mitigates the effects of sensor errors but still preserves the ability to recognize a task again. We accomplish this by defining special events for the set of events Λ in the DST model. For each task τ an event λ will be defined in the DST model that will be activated by the rule engine when the task is set to *done*. Each task τ is connected with its event λ through a doubting evidence: $\forall \tau \in T : \exists! \lambda \in \Lambda \wedge \exists! e \in E : s(e_{\tau,\lambda}) = false$.

4 Demonstration

We want to demonstrate the strengths of our combined approach by comparing it with Honecker's pure DST. For the comparison, we have decided to choose an installation of an electricity meter into a fuse box as a scenario for a maintenance task. This scenario has a practical use, because in the near future many old electricity meters in Germany have to be replaced by new smart meters due to the regulations (Messstellenbetriebsgesetz). This scenario would also be well suited for training purposes, as the capacity of trained technicians may not be sufficient to carry out the required number of modifications. In this case new trainees have to be hired, that could be prepared for this scenario with our combined approach.

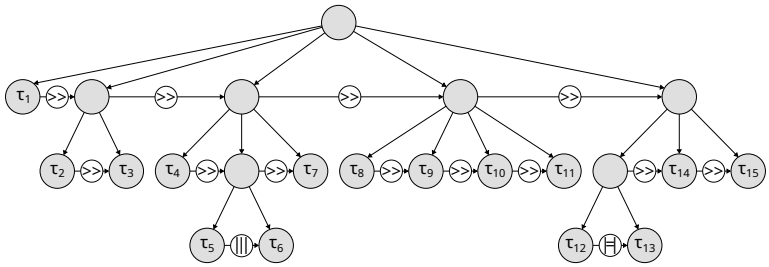
At a first step we turned the task "install electricity meter" into a DCoS-XML model by

1. breaking tasks down into smaller and smaller tasks up to the elementary tasks to create a hierarchical task structure
2. defining temporal constraints between tasks

The resulting DCoS-XML model is depicted in Figure 5. The outcome of the decomposition into a hierarchical structure are seven parent tasks and fifteen elementary tasks $\{\tau_1, \tau_2, \ldots, \tau_{15}\}$ encoded in the leaf nodes of the resulting DCoS-XML model.

Figure 5 also shows the temporal constraints between tasks that we identified after the decomposition. We identified 3 types of temporal operators between tasks. *Enabling* operators (denoted as ⋙) occur twelve times and are predominant. What is left is one *interleaving* operator (denoted as ⫼) between the tasks τ_5 and τ_6 and one *independence* operator (denoted as ⩦) between the tasks τ_{12} and τ_{13}.

Enabling operators indicate that the left task must be *done* before the right task can become *active*. Some *enabling* operators express a physical precondition.



Task	Name	Description
τ_1	open fuse box	In order to start the work, the door of the fuse box must be opened.
τ_2	check main fuse state	Before working on electricity parts, the main fuse state has to be checked.
τ_3	deactivate main fuse	To increase safety, the main fuse must be turned off.
τ_4	position meter cross	The meter cross is the attachment for the electricity meter and has to be positioned.
τ_5	fix meter on cross	The electricity meter is fixed on the meter cross.
τ_6	attach cables to meter	Fuse box's power cables are attached to the electricity meter.
τ_7	activate main fuse	Main fuse has to be turned on to test the installation.
τ_8	check voltage	The Cabling is tested with an two-poled voltage tester.
τ_9	re-deactivate main fuse	Main fuse must be off.
τ_{10}	attach cover	The electricity meter has a cover for shielding, which is attached.
τ_{11}	attach seals	To protect the counter from tampering, the cover of the electricity meter is sealed.
τ_{12}	take pictures of meter	For documentation purposes, a photo of the built-in electricity meter is shot.
τ_{13}	scan 2D code	For documentation purposes, the barcode of the built-in electricity meter is scanned.
τ_{14}	send report	Image and barcode are sent to the head office in a report.
τ_{15}	close fuse box	All tasks are finished and fuse box s closed

Fig. 5. Resulting DCoS-XML model of task "install electricity meter" and description of elementary tasks encoded in the leaf nodes of the DCoS-XML model

For example, a technician must complete task τ_1 (open fuse box) before he is even physically able to start task τ_2 (check main fuse), because the main fuse is inside the fuse box and cannot be seen if the door is closed. Other *enabling* operators express not necessarily physical conditions but rather reasonable orders. For example, for his own safety a technician should complete task τ_3 (deactivate main fuse) before he starts working on task τ_4 (position meter cross) where he could come in contact with electricity parts.

The *interleaving* operator between the tasks τ_5 (fix meter on cross) and τ_6 (attach cables to meter) states that a technician can do both tasks in parallel. Until the one of the two tasks are *done* he can even switch from the one task to the other task and back in an unlimited way. The reason is that both tasks produce interim results. Fixating the electricity meter onto the meter cross has three interim results: each is a screw, which connects the electricity meter with the meter cross, screwed on. Attaching cables to the electricity meter has six interim results - each is a cable attached to the electricity meter. As a technician is physically able to do both tasks at the same time he can produce the interim results in any order.

The *independence* operator between the tasks τ_{12} (take picture of meter) and τ_{13} (scan 2D code) is more restrictive than the *concurrent* operator. It states that a technician must complete one of the two tasks before he can start the other task, but can freely choose in which order the tasks are performed. Taking

a picture and scanning a 2D code both produce a single result - the picture and the decoded 2D code respectively. So a technician cannot start taking a picture, switch to scanning the 2D code and switch back to the same state of taking a picture that he left when he switched to scanning the 2D code because both task produce no interim results that could be continued.

At the second step we created the DST model. First we took the 15 elementary tasks from the DCoS-XML model and defined them in the DST model. Then we decided which events are important for recognizing the tasks. As event sources we used eye tracking and tool- and hand position recognition. For each task we decided which events support the task and connected the task with the appropriate event through a supporting evidence with an estimated mass. For example, the event λ "using wrench" is connected with the task τ_5 (fix meter on cross) through a supporting evidence $e_{\tau_5,\lambda}$ and is assigned a high mass $m_e = 0.9$ because a technician uses a wrench for only this task. We also decided for each task, which events may doubt a task and connected the task with such an event through a doubting evidence with an estimated mass. For example, when a technician tests voltage he uses a two-poled voltage tester. As it makes no sense to use the two-poled voltage tester in other tasks than task τ_{13} (scan 2D code) we assigned those tasks a doubting evidence connecting the task with the event "using two-poled voltage tester".

Fig. 6. The fuse box we use for our demonstration: On the left side the door of fuse box is closed. The fuse box's initial state with opened door is depicted in the middle and the fuse box's state of an installed electricity meter is shown on the right side

For our demonstration we use a real fuse box sponsored by ABB as presented in Figure 6. On the left side of that figure the fuse box's door is closed. The fuse box's initial state with opened door is depicted in the middle and the fuse box's state of an installed electricity meter is shown on the right side.

Before we started to perform the task we first decided in which order we execute the 15 elementary tasks. We have stuck to it, as demonstrated by a regional energy provider in Oldenburg. Then one of us would execute the tasks in this order while recording it with an eye tracker. The execution of the task lasts approximately five to six minutes. Afterwards we annotated the recording with tools in use and hand position data. Finally we applied task recognition to the annotated recording twice - once by using Honecker's simplified DST approach and once by using our combined approach. Both task recognitions were done with the same DST model. In this way we were able to estimate how our combined approach performs in comparison to Honecker's simplified DST approach.

The comparison is depicted in Figure 7. The three horizontal bars illustrate the tasks at a specific time by different colors. Bar ② shows the tasks, as they were done in reality. Bar ① and bar ③ show the tasks recognized by Honecker's approach and our combined approach respectively.

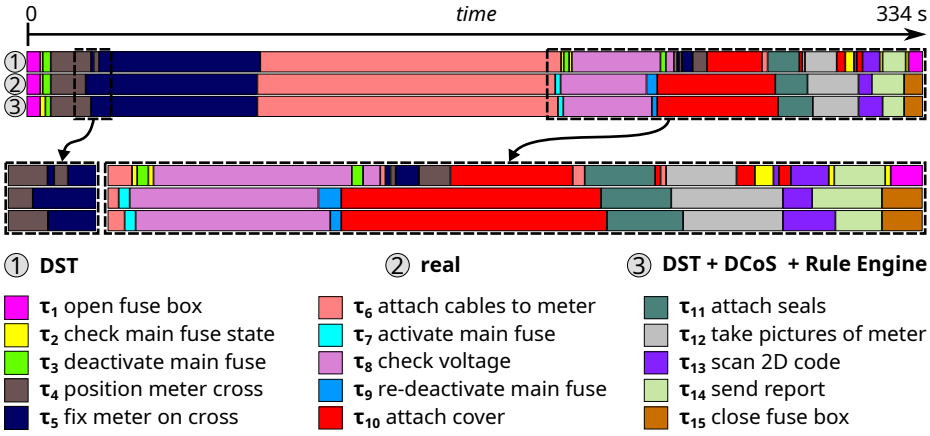


Fig. 7. Comparision of Honecker's DST with our combined approach: The three horizontal bars illustrate the tasks at a specific time by different colors. Bar *2* shows the tasks, as they were done in reality. Bar *1* and bar *3* show the tasks recognized by Honecker's approach and our combined approach respectively.

As can be seen in the figure, we were not able to recognize the correct order of the tasks with Honecker's approach, although we achieved a relative good accuracy of 0.826 for correctly recognized tasks. For example, as shown in the figure, task τ_1, τ_3, τ_6, τ_5 and τ_4 are recognized in between while actually task τ_{10} is being performed. The reason for this is that the evidences of these tasks are

very similar. A technician would use the same tool, would roughly look at the same area within the fuse box and would use his hands in the approximately same position while performing one of these tasks. Now if, for example, the technician's gaze deviates, a different task than the actual one could be recognized. The result reflects our initial guess that Honecker's simplified DST alone is not well suitable for strongly sequential tasks.

In contrast, with our combined approach we were able to recognize all tasks in the correct order with an accuracy for correctly recognized tasks of 0.958. Among other things, the accuracy is not 1 because the transition from one task to the other is difficult to detect. Often, newly started tasks can only be detected with a certain delay. The delay is due to the fact that the events of the evidences do not always occur at the beginning of a task. For example, when starting task τ_5 (fix meter on cross) a technician first takes a screwdriver and then, only after some time, he looks at the meter cross.

5 Discussion

In this paper we showed how to combine Honecker's simplified DST inferring with a rule engine operating on a DCoS-XML model to improve task recognition for strongly sequential tasks like maintenance tasks. Although our first attempts confirm this improvement, we still see room for improvement. The first improvement would be a more precise interpretation of the belief triplets Θ that are produced by the DST inference engine. Honecker's interpretation in a highly parallel helicopter scenario is to select those tasks as part of the pilot's current activity that have a belief of $p_\tau > q_\tau + r_\tau$ [14]. We instead generate belief values from the belief triplets to define an order relation on the set of tasks. Because only one task can currently be processed in a sequential maintenance task, we choose the task τ having the highest belief value β_τ. What we do not consider is the ignorance r_τ that expresses the uncertainty whether a task τ is processed or not processed. We suspect that a belief value β_τ supports or doubts a task τ with a greater probability the smaller the uncertainty r_τ is. In the future we want to investigate this.

Another improvement relates to recognizing the next task. We already reduce the probability of recognizing a completed task again. But we do not influence the probability of recognizing future tasks. The idea is to increase the probability for those tasks that are allowed to be processed after the current task. We want to experiment with these probabilities, so that it is more likely to recognize a valid next task but still possible to detect sequence violations.

References

1. Smart Maintenance für Smart Factories: Mit intelligenter Instandhaltung die Industrie 4.0 vorantreiben. acatech POSITION, Utz Verlag GmbH (2015)
2. Barnett, J.A.: Computational methods for a mathematical theory of evidence. In: Classic Works of the Dempster-Shafer Theory of Belief Functions (1981)

3. Becker, W., Brinkmann, F.: Kostenrechnung für die Instandhaltung: Ergebnisse einer empirischen Untersuchung. Bamberger betriebswirtschaftliche Beiträge, Otto-Friedrich-Univ. (2000)
4. Card, S.K., Moran, T.P., Newell, A.: The Psychology of Human-Computer Interaction. Erlbaum, Hillsdale (1983)
5. Casale, P., Pujol, O., Radeva, P.: Human activity recognition from accelerometer data using a wearable device. In: Proceedings of the 5th Iberian Conference on Pattern Recognition and Image Analysis. pp. 289–296. IbPRIA'11, Springer-Verlag, Berlin, Heidelberg (2011)
6. Charniak, E.: Bayesian networks without tears: Making bayesian networks more accessible to the probabilistically unsophisticated. AI Mag. **12**(4), 50–63 (Nov 1991)
7. Dempster, A.P.: Upper and lower probabilities induced by a multivalued mapping. Ann. Math. Statist. **38**(2), 325–339 (04 1967)
8. Diaper, D., Stanton, N. (eds.): The Handbook of Task Analysis for Human-Computer Interaction. CRC Press (2004)
9. Donath, D.: Verhaltensanalyse der Beanspruchung des Operateurs in der Multi-UAV-Führung. dissertation, Universität der Bundeswehr München (2012)
10. Donath, D., Schulte, A.: Behavior based task and high workload determination of pilots guiding multiple uavs. Procedia Manufacturing **3**, 990 – 997 (2015), 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015
11. Fahssi, R., Martinie, C., Palanque, P.: Embedding explicit representation of cyber-physical elements in task models. In: IEEE International Conference on Systems, Man and Cybernetics (SMC 2016) (2016)
12. Honecker, F., Schulte, A.: Evidenzbasierte Pilotentätigkeitserkennung unter Berücksichtigung unterschiedlich zuverlässiger Beobachtungen. In: 57. Fachausschusssitzung Anthropotechnik der DGLR: Kooperation und kooperative Systeme in der Fahrzeug- und Prozessführung. pp. 115–130. Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Obert e.V., Bonn (2015)
13. Honecker, F., Schulte, A.: Konzept für eine automatische evidenzbasierte Online-Pilotenbeobachtung in bemannt-unbemannten Hubschraubermissionen (2015)
14. Honecker, F., Schulte, A.: Automated online determination of pilot activity under uncertainty by using evidential reasoning. In: Harris, D. (ed.) Engineering Psychology and Cognitive Ergonomics: Cognition and Design. pp. 231–250. Springer International Publishing, Cham (2017)
15. Honecker, F., Schulte, A.: Kognitive und Workload-adaptive Unterstützung von Hubschrauberpiloten in multi-UAV Missionen (2017)
16. Huynh, T., Schiele, B.: Analyzing features for activity recognition. In: Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-aware Services: Usages and Technologies. pp. 159–163. sOc-EUSAI '05, ACM, New York, NY, USA (2005)
17. Kolekar, M.H., Dash, D.P.: Hidden markov model based human activity recognition using shape and optical flow based features. In: 2016 IEEE Region 10 Conference (TENCON). pp. 393–397 (Nov 2016)
18. März, M., Blechschmidt, N., Weck, D.L.: Projektstudie Chemie und Pharma: Wertorientiertes Instandhaltungs- und Asset Management. Tech. rep., ConMoto Consulting Group GmbH (2014)
19. Osterloh, J.P., Bracker, H., Müller, H., Kelsch, J., Schneider, B., Lüdtke, A.: Dcosxml: A modelling language for dynamic distributed cooperative systems. In: Proceedings of the 2013 11th IEEE International Conference on Industrial Informatics (INDIN). IEEE (7 2013)

20. Paternò, F.: Concurtasktrees: An engineered notation for task models. In: Diaper, D., Stanton, N. (eds.) The Handbook of Task Analysis for Human-Computer Interaction. Lawrence Erlbaum Associates, Mahwah (2004)
21. Rabiner, L.R.: Readings in speech recognition. chap. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pp. 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1990)
22. Schulte, A., Donath, D., Honecker, F.: Human-system interaction analysis for military pilot activity and mental workload determination. In: Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on (2015)
23. Stanton, N.A.: Hierarchical task analysis: Developments, applications, and extensions. Applied Ergonomics **37**(1), 55 – 79 (2006), special Issue: Fundamental Reviews
24. Yamato, J., Ohya, J., Ishii, K.: Recognizing human action in time-sequential images using hidden Markov model. In: Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on. pp. 379–385 (1992)