Adapting Workflows to Intelligent Environments

Melanie Hartmann*
AGT Group (Germany) GmbH
Darmstadt, Germany
mhartmann@agtgermany.com

Marcus Ständer
Department of Computer Science
Telecooperation Group
Technische Universität Darmstadt, Germany
marcus@tk.informatik.tu-darmstadt.de

Victoria Uren
Department of Computer Science
OAK Group
University of Sheffield, UK
v.uren@dcs.shef.ac.uk

***Abstract*—Intelligent environments aim at supporting the user in executing her everyday tasks, e.g. by guiding her through a maintenance or cooking procedure. This requires a machine processable representation of the tasks for which workflows have proven an efficient means. The increasing number of available sensors in intelligent environments can facilitate the execution of workflows. The sensors can help to recognize when a user has finished a step in the workflow and thus to automatically proceed to the next step. This can heavily reduce the amount of required user interaction. However, manually specifying the conditions for triggering the next step in a workflow is very cumbersome and almost impossible for environments which are not known at design time. In this paper, we present a novel approach for learning and adapting these conditions from observation. We show that the learned conditions can even outperform the quality as conditions manually specified by workflow experts. Thus, the presented approach is very well suited for automatically adapting workflows in intelligent environments and can in that way increase the efficiency of the workflow execution.**

I. Introduction

We are dealing with an ever increasing number of sensors in our everyday life which provide information about the current context. This enables intelligent environments to be better aware of the user's current actions and thus to better support the user in executing tasks. A task can thereby refer to the usage of a single product, e.g. repairing a dishwasher or descaling a coffee machine, as well as to the execution of a task involving several products, e.g. cooking with a smart oven, a smart pot etc. If the user is not familiar with a task, she usually has to refer to written instructions or executes the task the way she thinks is best. Both approaches hamper the quality of the execution. Therefore, the products or the whole environment should be able to guide the user in executing tasks. This requires a machine processable specification of the tasks stating the steps which are required to accomplish them. The task descriptions should also contain information about when to proceed to the next step to avoid that the user has to press a "next" button all the time. Workflows have proven an efficient means for such a representation (see [1] or [2]). However, specifying these workflows is quite tedious, especially stating which (context) events should trigger the next step. Moreover, it is usually not possible to specify all possible triggers at design time as the concrete environment in which the task will be executed (including the available sensors) is usually unknown (e.g. the user's kitchen). For that reason, the workflow should be able to automatically adapt to new environments.

In this paper, we present a novel approach to augment an existing workflow description with information when to proceed to the next step for a given environment. For that purpose, we rely on observing the user in executing the workflow in this environment. Our approach is able to deal with very few training data (two runs already provide good results) which is important when being applied in practice. Currently, we do not automatically perform adaptations and only suggest them to the user, because we want to avoid the user feeling that she has lost control of the system. However, the results achieved by the presented approach are very promising so that a complete automation is conceivable.

We show that the approach achieves results comparable to those of workflow experts with respect to (i) the ability to recognize the end of an activity, and (ii) the meaningfulness of the proposed triggers. For this evaluation, we generated a novel dataset containing recordings of the execution of a workflow in an intelligent kitchen. The dataset can be downloaded from our website and thus hopefully supports further research on workflows in intelligent environments.

In the following, we at first describe how workflows for intelligent environments are represented (Section II). Then, we introduce our approach, i.e. which types of context triggers it supports (Section III) and how it learns context triggers from observation (Section IV). In Section V, we report on the results of our evaluation. Finally, we give an overview of related approaches (Section VI) and outline further research (Section VII).

II. Workflows in Intelligent Environments

Workflows can be seen as graph structures with nodes (*activities*) and edges (*transitions*). Executing a workflow means following transitions from at least one start node and executing all activities on the way. Thereby, conditions can be specified, expressing when an activity is finished and thus when its outgoing transition should be followed (*triggering conditions*). An activity can have several outgoing and incoming transitions, which can result in very complex workflow structures. However, we focus here on purely sequential workflows (with one incoming and one outgoing transition) and will extend

*The author was employed at the Technische Universität Darmstadt when conducting the research presented in this paper.

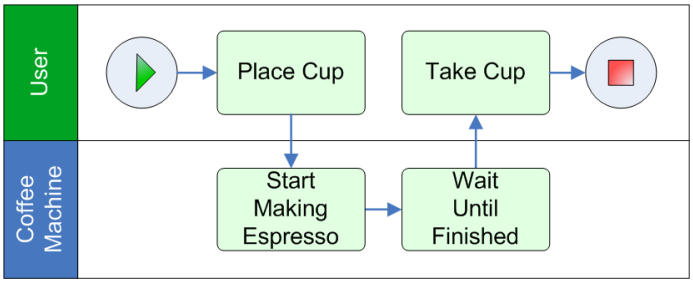

Fig. 1. Example workflow for preparing espresso

it to other workflows in future work. Sequential workflows have the advantage that they can be learnt more reliably and are sufficient for many real world applications. Figure 1 shows a short sequential example workflow for preparing an espresso consisting of four activities and three transitions. Thereby, two activities (*"Place Cup"*) and (*"Take Cup"*) need to be performed by the user whereas the other two can be automatically executed by a coffee machine.

Triggering conditions can be expressed with the help of context information. Triggering allows a change from explicit interaction, e.g. an acknowledgement via a user interface, to implicit interaction. For example, the end of the *"Place Cup"* activity in our example workflow could be automatically recognized by RFID events, if an RFID tag is attached to the cup and an RFID reader to the coffee machine: As soon as the user places the cup under the drain of the coffee machine, the coffee machine could automatically brew the espresso without requiring the user to press a button.

III. Triggering Conditions In Workflows

As stated before, for each activity in a workflow a condition can be specified that needs to be satisfied in order to proceed to the next activity, i.e. a triggering condition. These conditions usually refer to the current context. In this section, we will point out how we represent context and which types of conditions are supported by our approach.

We represent **context information** by its type and a list of attributes Att in form of key value pairs (k_i, v_i) which represent the actual content. Further, a timestamp t can be attached to it. We thus define context information c as

$$c = (type, Att = \{(k_i, v_i)\}, t)$$

For example, temperature information can be represented as $c = ('temp', \{('unit', 'C'), ('value', 21.4)\}, 145)$.

A triggering condition at first needs to state to which type of context information it refers. Further, it should be possible to specify restrictions on the actual content of the context information, e.g. that the temperature is more than 25°C. For that purpose, we define a **basic condition** (or rule) as:

$$r = (type, key, op, value)$$

with key referring to one of the attributes k_i of context information, op being the operator of the condition and $value$ the value to compare to. A condition for a temperature of more than 25°C could thus be expressed as $('temp', 'value', \geq, 25)$. Our approach supports the following set of operators but many more are conceivable:

$$op \in \{=, \leq, \geq, increase, decrease, change\}$$

Thereby, $increase$, $decrease$ and $change$ refer to a change of the attribute's value with respect to its value at the beginning of the activity. For example, $('temp', 'value', increase, 6)$ means that the temperature value has to rise by more than 6°C in the course of the activity to meet the condition. Whether context information c satisfies a condition r is given by[1]:

$$sat(c, r) \Leftrightarrow \ type(c) = type(r) \wedge \\ \exists (k, v) \in Att(c).(k = key(r) \wedge f(v, op(r), value(r)))$$

Thereby, $f : (x, op, y) \mapsto \{true, false\}$ is a boolean function which is defined for each operator op and two values x, y as follows:

- $f(x, =, y) \Leftrightarrow x = y$
- $f(x, \leq, y) \Leftrightarrow x \leq y$
- $f(x, \geq, y) \Leftrightarrow x \geq y$
- $f(x, increase, y) \Leftrightarrow x - v_{init} \geq y$, with v_{init} being the initial value of the corresponding attribute at the beginning of the activity
- $f(x, decrease, y) \Leftrightarrow v_{init} - x \geq y$
- $f(x, change, y) \Leftrightarrow |x - v_{init}| \geq y$

The next activity in the workflow is then triggered as soon as context information is observed that satisfies the *basic condition* r, i.e.

$$satisfied(r, C) \Leftrightarrow \exists c \in C.sat(c, r)$$

with C being the set of observed context information.

Sometimes these *basic conditions* are not sufficient and we need to specify a temporal sequence of conditions $(r_1 \rightarrow ... \rightarrow r_n)$ or to combine several conditions $(r_1 \wedge ... \wedge r_n)$. For example, $('temp', 'unit', =, 'C') \wedge ('temp', 'value', \geq, 25)$ ensures that we use a sensor that reports the temperature in degree Celsius.

Temporal conditions In order to make a statement about temporal relationships, we need to know when the context information that met a condition was observed, i.e. we need to consider $t(c)$. A temporal composition of conditions is then satisfied if:

$$satisfied(r_1 \rightarrow ... \rightarrow r_n, C) \Leftrightarrow \exists \{c_i \in C | i = 1..n\}. \\ (t(c_1) \leq ... \leq t(c_n)) \wedge sat(c_1, r_1) \wedge ... \wedge sat(r_n, c_n)$$

Combined conditions: A *combined condition* is satisfied as soon as context information was observed that satisfies all the conditions it is composed of, i.e.

$$satisfied(r_1 \wedge ... \wedge r_n, C) \Leftrightarrow \exists c \in C.sat(c, r_1) \wedge ... \wedge sat(c, r_n)$$

Finally, we might want to specify several **alternatives** that can trigger the next activity. For that purpose, workflows allow

[1]For ease of readability, we will use the abbreviation $a(x)$ to refer to the first parameter of $x = (a, b, c)$ etc.

us to specify several conditions per activity, thus alternatives (or the $\vee$ operator) are implicitly defined. We refer to the set of all conditions for an activity a as R_a. The next activity is automatically triggered as soon as one condition is satisfied, i.e. $\exists r \in R_a.satisfied(r, C)$.

IV. Learning Triggering Conditions

In this paper, we present a novel approach for learning triggering conditions from observation and thus for disburden the user or developer from having to manually specify them. At first, we need to track how a user executes a workflow. The end of an activity and thus the start of the next activity can either be determined based on (i) a context-trigger already specified for the activity, or (ii) the user who tells the system to move on to the next activity, e.g. by pressing a button. This data is stored along with all context information which incurred during the execution of the workflow in a log L_i (one log per workflow execution).

As stated before, context information consists of a type, attributes and a timestamp, i.e. $t(c)$. The execution of an **activity** a is characterized by a start (t_{start}) and end time (t_{end}), and a set of initial context information C_{init}, i.e.[2]

$$a = (t_{start}, t_{end}, C_{init})$$

C_{init} contains all initial values v_{init} which are required for determining how the value of an attribute changes in the course of an activity (e.g. that the temperature increases by 10°C). C_{init} is defined as follows:

$$\begin{aligned} C_{init} = &\{c \in L_{<a} | \forall c_i \in L_{<a} \backslash \{c\}. \\ &type(c_i) = type(c) \Rightarrow t(c_i) \leq t(c)\} \end{aligned}$$

with $L_{<a}$ being the set of context information that was observed before a started, i.e. $L_{<a} = \{c \in L_i | t(c) \leq t_{start}(a)\}$.

Further, we define the set of context information recorded during the execution of a in L_i as

$$C_a = \{c \in L_i | t_{start}(a) \leq t(c) \leq t_{end}(a)\}$$

The logged data is then used to learn triggering conditions from observation. We require at least two runs, i.e. logs, to be able to obtain reliable results, because it is hardly possible to distinguish between noise and relevant context data from only one run. With our approach, two logs are usually already sufficient to achieve good results as we will show in the evaluation. For learning triggering conditions, we at first determine possible conditions for each activity per log (Section IV-A) and then combine the conditions from the different logs (Section IV-B). Finally, we rate the conditions according to some quality metrics to identify the most relevant conditions for triggering the next activity (Section IV-C).

[2]Formally correct, we would need to explicitly state the log file from which the information was obtained, e.g. $a(L_i)$ or $C_a(L_i)$, but we will abbreviate it as a or C_a to ease the readability if it is clear from the context to which log file it refers.

A. Calculating Conditions for a Single Log $R_a(L_i)$

At first, we iterate over all activities stored in a single log L_i and determine the set of conditions $R_a(L_i)$ that could be relevant triggering conditions (see Algorithm 1). R_a consists of '='-conditions for all possible attributes of the context information that occurred in the course of the execution of the activity. For example, $('temp', 'unit', =, 'C')$ for the attribute $'unit'$. Further, a '$\leq$'- and a '$\geq$'-condition for each context attribute with its minimal and maximal value, respectively (e.g. $('temp', 'value', \leq, 70)$). For the other three operators, also one condition is computed per operator and context attribute with the maximal 'increase', 'decrease' or 'change'. From all observed combinations of attribute values we built corresponding *combined conditions*. In order to limit the amount of resulting conditions and their complexity, we only consider *combined conditions* consisting of *basic* '='-conditions. *Temporal conditions* are only determined in the next step when combining the results of different logs.

Algorithm 1 Calculate R_a for log L_i

Definition: $amountAttributes(tp)$: amount of attributes for context information of type tp, e.g. $tp = 'temp'$ has two attributes ($'unit'$, $'value'$) in our example
Definition: b: buffer $\in [0, 1]$
for all $tp \in \{type(c) | c \in C_a\}$ **do**
  // basic conditions
  for $1 \leq x \leq amountAttributes(tp)$ **do**
    for all $y \in \{v_x(Att(c)) | c \in C_a \wedge type(c) = tp\}$ **do**
      add $(tp, k_x, =, y)$ to R_a
    end for
    $v_{init} = v_x(Att(c))$ with $c \in C_{init}(a) \wedge type(c) = tp$
    $min = \min\{v_x(Att(c)) | c \in C_a \wedge type(c) = tp\}$
    $max = \max\{v_x(Att(c)) | c \in C_a \wedge type(c) = tp\}$
    $absMax = \max\{v_{init} - min, max - v_{init}\}$
    if $((1-b) \cdot min < v_{init})$ **then**
      add $(tp, k_x, \leq, (1-b) \cdot min)$ to R_a
    if $(min < v_{init})$ **then**
      add $(tp, k_x, decrease, (1-b) \cdot (v_{init} - min))$ to R_a
    if $((1-b) \cdot max > v_{init})$ **then**
      add $(tp, k_x, \geq, (1-b) \cdot max)$ to R_a
    if $(max > v_{init})$ **then**
      add $(tp, k_x, increase, (1-b) \cdot (max - v_{init}))$ to R_a
    if $(absMax > 0)$ **then**
      add $(tp, k_x, change, (1-b) \cdot absMax)$ to R_a
  end for
  // Combined Conditions
  for all $c \in \{c \in C_a | type(c) = tp\}$ **do**
    $r_\wedge = true$
    for all $(k, v) \in Att(c)$ **do**
      $r_\wedge = r_\wedge \wedge (tp, k, =, v)$
    end for
    add $r_\wedge$ to R_a
  end for
end for

The calculated conditions are filtered by checking whether they "fit" to the corresponding initial value v_{init}. For example, if $v_{init} = 5$ and the minimally observed value is 6, the value did not really decrease, so $(x, y, \leq, 6)$ (for corresponding context attribute y of type x) is not considered relevant.

Humans do not use the minimal/maximal value for specifying conditions and rather leave a buffer to this maximum value (which could also be observed in our evaluation). For example, if the maximally observed value is 10, a human usually states a condition with a lower value, e.g. $(x, y, \geq, 8)$. We try to mimic this behavior in our approach and also apply a buffer $b \in [0, 1]$ to the minimally/maximally observed values by multiplying them with $(1 - b)$. Which value should be chosen for b will be discussed in the evaluation (Section V).

B. Combining Conditions from Several Logs

In the next step, the conditions of the single logs $R_a(L_i)$ are combined to build the final set of conditions $R_a(\mathcal{L})$ with $\mathcal{L} = \{L_i\}$ being the set of all logs. For that purpose, we determine the best conditions for every combination of context type and attribute's key (tp, k). $R_a(\mathcal{L})$ contains all *combined* and '='-conditions of the single logs. From all other conditions, only the one with the maximal value from all logs (for '$\leq$'-conditions) or the minimal value (for all other conditions) is maintained, i.e.

$$\begin{aligned} R_a(\mathcal{L}) = \bigcup_{tp,k} \{ & \{(tp, k, =, x) \in R_a^{\cup}\} \\ & \cup \{r \in R_a^{\cup} \wedge isCombined(r) \wedge type(r) = tp\} \\ & \cup \{(tp, k, \leq, \max\{x | (tp, k, \leq, x) \in R_a^{\cup}\})\} \\ & \cup_{op \in O} \{(tp, k, op, \min\{x | (tp, k, op, x) \in R_a^{\cup}\})\}\} \end{aligned}$$

with $R_a^{\cup} = \bigcup_i R_a(L_i)$, $isCombined(r) \Leftrightarrow r$ is a *combined condition*, and $O = \{\geq, increase, decrease, change\}$.

Next, we identify relevant *temporal conditions*. In order to keep the complexity of conditions low, we restrict *temporal conditions* to two steps referring to the same context type, e.g. increase and decrease of the temperature. Further, not all combinations of *temporal conditions* are sensible, e.g. $(x, y, \geq, 5) \rightarrow (x, y, \geq, 10)$ is not really useful, because it can usually be replaced by $(x, y, \geq, 10)$. For that reason, we restrict the *temporal conditions* to combinations of '$\leq$', '$\geq$' and of '$increase$','$decrease$' conditions. Moreover, we also allow the combination of two *combined conditions* with different content (but of the same type). To compute the *temporal conditions*, we calculate for each relevant condition r_1 in $R_a(\mathcal{L})$ the corresponding r_2 on a subset $\mathcal{L}_{r_1}$ of the log $\mathcal{L}$. $\mathcal{L}_r$ contains all data that was recorded after the condition r was met, i.e. $\mathcal{L}_r = \bigcup_i \{x \in L_i | c = match(r, C_a(L_i)) \wedge t(x) \geq t(c)\}$ with $match(r, C_a(L_i))$ being the first context information in $C_a(L_i)$ that satisfies r (i.e. with $sat(c, r) = true$). Thus, the following *temporal conditions* are added to $R_a(L)$:

$$\begin{aligned} R_a(\mathcal{L}) = R_a(\mathcal{L}) \cup \bigcup_{tp,k} \{ & \{(r_1 \rightarrow r_2) | r_1 = (tp, k, op_1, x) \in R_a(\mathcal{L}) \\ & \wedge \exists r_2 = (tp, k, op_2, y) \in R_a(\mathcal{L}_{r_1}) \wedge \{op_1, op_2\} \in T\} \\ & \cup \{(r_1 \rightarrow r_2) | r_1, r_2 \in R_a(\mathcal{L}) \wedge r_1 \neq r_2 \wedge \\ & isCombined(r_1) \wedge isCombined(r_2)\}\} \end{aligned}$$

with $T = \{\{\leq, \geq\}, \{increase, decrease\}\}$.

C. Selecting Best Triggering Conditions

From the set of calculated conditions $R_a(\mathcal{L})$ we select those which most likely reflect the end of an activity. For that purpose, we determine the quality of each condition r according to several parameters[3]:

- **applicability** app: ratio of logs L_i in which the condition was satisfied during the execution of the corresponding activity a, i.e.
$$app(r, a) = \frac{\sum_i satisfied(r, C_a(L_i))}{|\mathcal{L}|}$$
This results in a value between 0 and 1, whereby 1 means that r was satisfied in all logs and 0 that it was not satisfied in any log.
- **specificity** $spec$ reflects how specific the condition is to an activity a, i.e. whether it is not also satisfied in the course of many other activities. This metric is important to filter conditions which are generated by noise. For example, if a sensor continuously reports changing data, the algorithm will generate some (irrelevant) conditions from it. However, these conditions are satisfied during the execution of many activities and can thus be filtered accordingly. We compute the specificity of a condition r as the ratio of other activities to which r could *NOT* be applied:
$$spec(r, a) = 1 - \frac{\sum_{a_i \in A \setminus \{a\}} app(r, a_i)}{|A| - 1}$$
with A being the set of all activities of the workflow. Thus, a specificity value of 1 indicates that r was only met during the execution of a. whereas a specificity of 0 means that the condition r was also satisfied during the execution of all other activities.
- **timing** dt (delta t): Moreover, it is important that the condition should be satisfied as close to the recorded end of the activity as possible, i.e. it should not already trigger the next activity if the user has just started the activity. We define this time gap dt for one log L_i as $dt(r, L_i) = t_{end}(a) - t(c)$ with $c = match(r, C_a(L_i))$. (As stated before, $match(r, C_a(L_i))$ refers to the first context information in $C_a(L_i)$ that satisfies r) We then define $dt(r)$ as the average of all those values, i.e
$$dt(r) = \frac{\sum_i dt(r, L_i)}{|\mathcal{L}|}$$

[3]For simplicity, we assume that $\mathcal{L}$ only consists of complete logs, i.e. logs that contain data for each activity of the workflow.

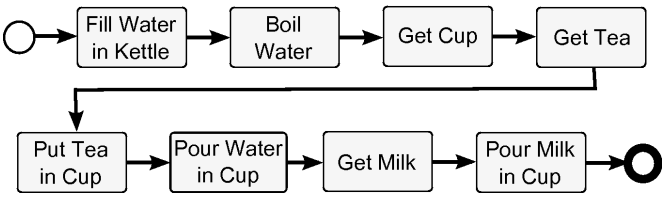

Fig. 2. Workflow for preparing tea with milk

We sort the conditions according to these parameters –with app having the highest priority and dt the lowest. As stated before, we currently only suggest these conditions to the user or workflow developer (via a workflow editor), but we aim to automatically include them in the workflow description in the future. To prevent that the user or workflow gets flooded with conditions that vary only slightly, e.g. $(x, y, \geq, z_1)$ and $(x, y, increase, z_2)$, we restrict the suggested conditions to the best condition per context type.

V. Evaluation

For evaluating the presented approach, we recorded a workflow for preparing tea in a kitchen equipped with several sensors. Further, we asked six workflow experts to fill out a questionnaire regarding the conditions they would use for annotating the workflow in that specific environment. In the following, we at first describe the setup of the experiment and the questionnaire. Then we evaluate our approach regarding *expressiveness*, *performance* and *meaningfulness*. Performance thereby refers to how well these conditions can predict the end of an activity. Meaningfulness captures the subjective ratings of human experts which does not necessarily reflect the performance of the conditions. For example, a very complex condition can yield good results, but would not be rated as appropriate by human judges. For evaluating the performance and meaningfulness of our automated approach, we compared the results to those of the human experts. We expected that the experts would outperform our approach regarding both metrics, because they can profit from background knowledge, e.g. which axis of a 3D-gyroscope is relevant for detecting movements in a specific direction, in contrast to the computer.

A. Experimental Setup

The workflow we chose for evaluating our approach is shown in Fig. 2. The kitchen used for the experiment with all items can be seen in Fig. 3. We attached RFID tags to all items and equipped some of them with Phidget[4] sensors: the tea caddy with vibration and light sensors, the cup with sound and temperature sensors, and the milk bottle with a gyro sensor. For the latter, we only recorded the average x-, y- and z-values for time windows of 2s in order to reduce the amount of recorded data. During the execution of the workflow, the user was wearing an RFID reader around his wrist. The computer, which was attached to all sensors and

Fig. 3. Kitchen and items used for experiments

the RFID reader, prompted the user which activity to execute. The user had to press a button in order to proceed to the next activity. We recorded 8 runs with two different users. As there is a time gap between the "ideal" and the recorded end of an activity (when the user presses the button), we also annotated in 4 of these datasets the time when we would consider an activity to be finished ($t_{idealEnd}(a)$). The recorded datasets can be downloaded from our website[5].

B. Questionnaire

The six workflow experts were given a description of the workflow and the experimental setup including graphs illustrating the values for each sensed context type for two randomly chosen runs. These graphs were also annotated with beginning and end of the different activities. The questionnaire consisted of two parts:

At first, we asked the experts to state which conditions they would choose for annotating the workflow for the given environment. For example, "a temperature rise to more than 80°C". Thereby, we did not restrict them in the type of conditions. These conditions were used to evaluate the expressiveness of our approach and to gather a "gold standard" for the performance and meaningfulness of the learned conditions.

Next, the experts should rate the suitability of different conditions for detecting the end of an activity in our experimental setup. Here, we limited the conditions to those that are supported by our approach and were only interested in the type of condition (e.g. $\leq$) and not in concrete values. We used a scale from 0 to 3 (with 0: "should not be used", 1: "Not best condition, but could detect the end of the activity in some cases", 2: "Should work in most cases / best choice with given sensors", 3: "Very well suited condition, should reliably detect the end of the activity"). The ratings were used to evaluate the meaningfulness of the different conditions.

C. Results

The experts stated 84 conditions in total, i.e. they specified on average 1.75 conditions per activity. 45% were *basic*

[4]http://www.phidgets.com

[5]http://www.smartproducts-project.eu/teaDataset

conditions making use of all the operations that are also supported by our approach. 30% were *combined conditions* each consisting of two *basic conditions*. The final 24% were *temporal conditions*, whereby all but one consist of two conditions.

Expressiveness 93% of all triggering conditions specified by the workflow experts can be represented with our approach. Five of the six conditions that are not covered, deal with "constant" values over a specific period of time to detect that a movement has stopped, e.g. that the gyro sensor does not report a value above 5 or below -5 for 4 seconds. The remaining not covered condition is a *temporal condition* consisting of three different *basic conditions* of two different types. Our approach could be easily extended to cover this condition, but we wanted to keep the resulting conditions as simple as possible and for that purpose restricted our approach to two-step *temporal conditions* of the same context type (e.g. a '$\geq$'- and a '$\leq$'-conditions for the temperature value).

This shows that our approach can cover most conditions required for representing triggering conditions in workflows, at least for such a simple example as used for the evaluation.

Performance In order to evaluate the performance of the generated conditions, we computed their *applicability* and *timing* and compared the results to those achieved by the experts.[6] We had to exclude one expert, as he did not state concrete numbers for the conditions. Further, we excluded the results for the "boil water" activity because there was no way to detect the end of this activity with the given sensors. We used a slightly modified version $dt'$of dt for the evaluation: (i) we normalized delta t according to the length of the corresponding activity to enhance comparability, and (ii) we used the annotated "ideal" ends of activities ($t_{idealEnd}$) instead of t_{end} to better reflect the desired outcome, i.e. we used

$$dt'(r, L_i) = \frac{t_{idealEnd}(a) - t(c)}{t_{idealEnd}(a) - t_{start}(a)}$$

As discussed before, we also try to mimic human behavior in applying a buffer to the maximum / minimum value for the conditions (see Section IV-A). We evaluated the effects of the chosen buffer b on the performance of our approach. We calculated app and dt' when considering only the best condition suggested by our approach, i.e. $n = 1$, using two randomly chosen datasets for training and all others for testing. Fig. 4 shows that an increasing buffer b improves the applicability app but worsens the timing (increasing dt') as was to be expected. As the best results are achieved with $b = 0.1$ we use this for the further evaluation.

Next, we computed the applicability and timing achieved with the conditions specified by the workflow experts. The six conditions that cannot be expressed with our system were simplified or ignored after consultation with the corresponding

[6]The third quality parameter *specificity* is only relevant for selecting conditions and does not give evidence about the performance.

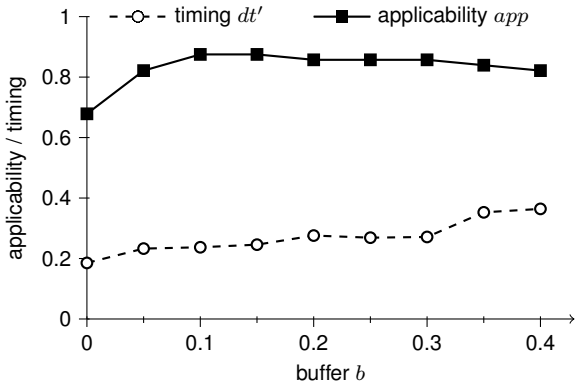

Fig. 4. Average applicability (app) and timing (dt') for increasing buffer b (only considering the top ranked condition generated by our approach, i.e. $n = 1$)

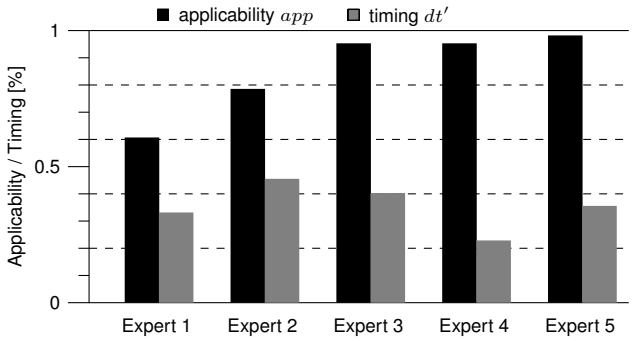

Fig. 5. Average applicability (app) and timing (dt') of the conditions stated by the different experts

experts. We used all datasets but the two used in the questionnaire for testing. Fig. 5 shows the performance per expert. It is very variable which shows that it is a difficult task for humans to define *the* best conditions for a workflow.

We compared the results to the one achieved by our generated conditions. The computer was trained on the same two datasets that were given to the experts. We computed the results when considering the best n conditions generated by our approach for $n = 1, 2, 3$ (as stated before, several condi-

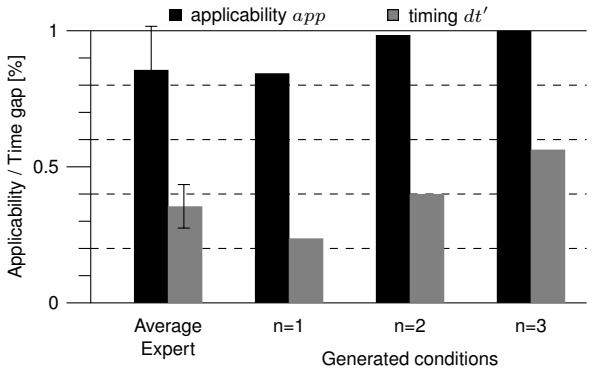

Fig. 6. Average applicability (app) and timing (dt') achieved by experts (with standard deviation) and the conditions generated by our approach for varying amount of considered conditions (with $b = 0.1$)

tions per activity are treated as alternatives). Fig. 6 shows the performance regarding applicability and timing compared to the average results achieved by experts. As to be expected, app and dt' increases with the number of considered conditions n. If we only take the top ranked condition ($n = 1$) into account, the applicability is comparable to the one achieved by the average expert (-1%), the corresponding timing is even better than the one of the average expert (i.e. smaller dt', -11%) which exceeded our expectations. With $n = 2$ (which corresponds approximately to the amount of conditions specified by experts per activity, i.e. 1.75) it is the other way round: The applicability achieved by our approach is higher (+13%), but the timing is slightly worse (+5%).

To sum it up, we could show that the generated conditions can keep up with conditions specified by human workflow experts with respect to their performance (i.e. their applicability and timing) and even outperform the results achieved by some experts.

Meaningfulness For computing the meaningfulness of conditions, we rated them according to the human judgements. Thereby, we again excluded the "boil water" activity. The agreement between the different experts regarding the best conditions was lower than expected: the interrater agreement κ (according to [3]) is 0.21 and the best condition per activity achieved only an average score of 1.9 (*rating per condition*). For gaining a gold standard, we computed the average score per activity for the conditions specified by the experts. For each activity and expert, we considered only the condition with the highest score. The average expert achieves a score of 1.5 (with standard deviation of 0.3) and our approach with $n = 2$ only a score of 1.2 (see Fig. 7). This is due to the fact that our approach often suggested conditions the experts did not consider. For example, our approach preferred conditions when the connection to the RFID tag is lost (and thus e.g. the cup released) while the users preferred conditions when the tag was read. When we discussed issues like this with the experts they often agreed that such a condition might detect the end of an activity better than those they had selected. This shows that it is hard to suggest the right condition every user would agree upon and that the conditions with the highest ratings are not necessarily the best ones.

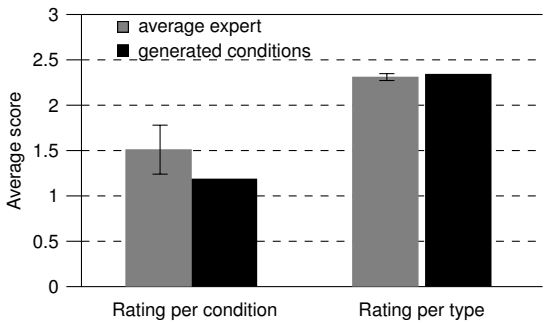

Fig. 7. Average rating per condition and type (considering the two top ranked generated conditions, i.e. $n = 2$) (with standard deviation for average expert)

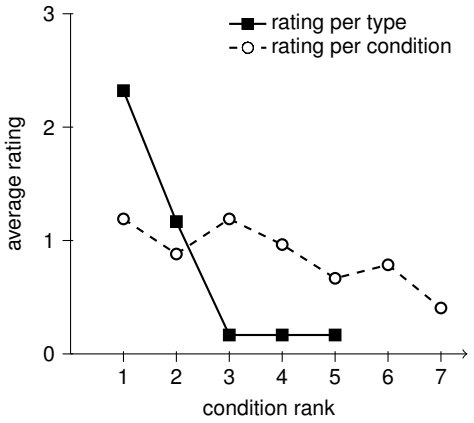

Fig. 8. Average rating per type and condition for the learned conditions ranked by computed relevance

The conditions that were rated highest often differed only slightly between the different users, e.g. one user rated $('temp','value', \geq, x)$ with the highest score while another preferred the $increase$ variant of this condition. For that reason, we also calculated ratings per sensor and RFID tag (*rating per type*) as we assumed that the experts can better agree on the relevance per type and could thus give us a better estimation of the meaningfulness of a condition. The score of a condition was calculated as the average of the maximum ratings the corresponding sensor / tag could achieve per expert, no matter which specific operation was rated. Here, the experts as well as our approach achieved an average rating of 2.3 with a much lower standard deviation between experts of 0.04 (see Fig. 7).

Finally, we evaluated whether the sorting of conditions by our approach reflects the relevance of the conditions. We computed the average ratings for the top ranked conditions as can be seen in Fig. 8. The solid line shows the average rating per type. You can see that the first condition is usually considered very relevant, the second as more or less relevant and the remaining conditions as hardly relevant. We also wanted to know whether the relevance of the exact type of condition (i.e. which operator is used etc.) is also reflected in the ranking of conditions.[7] The dashed line in Fig. 8 shows the results: the rating per condition also drops with increasing rank, but due to the problems arising with the rating per condition the trend is not as clear as the one for the rating per type.

To sum it up, the generated conditions are rated as meaningful as the conditions specified by human experts if we focus on the context type of the conditions (rating per type). Moreover, we showed that our approach successfully ranks the conditions based on their relevance according to the human judgements and thus meets the requirements for automatically adapting workflows in the future.

[7]For this purpose, we considered all conditions per type (in contrast to the general approach as discussed in Section IV-C).

VI. Related Work

In this section, we give an overview of work in three related areas: (i) context-aware workflow description languages, (ii) learning workflows for business processes, and (iii) learning workflows in intelligent environments.

In the literature, there exist several approaches how context information can be considered in workflows (e.g. [2], [4]). There also exist interesting contributions in the closely related domain of context-aware service composition, e.g. [5]. However, none of these approaches deals with learning or automatically adapting workflows.

In the area of business processes, there exists a plethora of work about mining workflow processes from logs (e.g. [6], [7], [8], [9]). However, their approaches cannot be directly applied to intelligent environments, e.g. because activities are harder to identify in intelligent environments than in business processes. Further, these approaches usually only cover mining of the workflow structure ("Graph mining", for an overview see [10]) and not learning conditions for triggering the next activity. Approaches which deal with the problem of mining triggering conditions (e.g. [8]) make use of decision trees which are for example not able to express temporal relations in conditions (which is important in intelligent environments as could be seen in the evaluation) and are not well suited for dealing with the multitude of highly dynamic context events.

Learning structures of tasks in the area of intelligent environments focuses usually on the aspect of using this information for activity recognition (e.g. building HMMs). For example, Kasteren et al. [11] learn the user's activity from low level sensor readings and Youngblood and Cook describe in [12] an algorithm for learning hierarchical models to predict the user's behavior in order to automate tasks in a smart home. Their results can be used as input (i.e. context) for our approach. However, none of these approaches uses explicit workflow models and can be used to support the user in executing a task.

One approach that learns workflows from observation and uses this data to support subsequent executions of the workflow was presented by Schneider [13]. He introduces the *Semantic Cookbook* which allows users to record videostreams of cooking procedures. The user can specify breakpoints during recording that specify states in the kitchen which have to be met during playback in order to proceed with the cooking procedure. However, the *Semantic Cookbook* is tailored to the cooking domain and a specially equipped kitchen and can thus not be easily applied to arbitrary workflows like our approach.

VII. Summary and Outlook

In this paper, we presented a novel approach for automatically adapting workflows to intelligent environments. For that purpose, we learn conditions to detect the end of an activity based on the current context. This enables us to automatically proceed to the next activity in a workflow and thus to heavily reduce the required user interaction. We compared the learned triggering conditions to those specified by workflow experts. We showed that the quality of the generated conditions is comparable to that achieved by workflow experts (wrt. performance and meaningfulness). Thus, our approach is able to relieve the user or workflow developer from manually specifying the conditions without hampering the quality of the workflow and can in that way simplify the adaptation process to a great extend.

As stated before, our final aim is to automatically adapt workflows to intelligent environments with none or at least minimal user involvement. However, this most probably has an effect on the perceived usability of the system, because of the loss of control for the user. For that reason, some more studies need to be performed how this effect can be reduced and/or how workflows (or the suggested modifications) can be represented in a way that can be easily understood by end-users.

Moreover, we want to learn those conditions used by the workflow experts which are not yet covered by our approach (that a context value does NOT change for a given period of time). We also want to extend our approach to workflows that are not purely sequential and learn whole workflows from scratch in intelligent environments.

Acknowledgments This research was conducted within the SmartProducts project funded as part of the Seventh Framework Programme of the EU (grant number 231204).

References

[1] C. Kunze, S. Zaplata, and W. Lamersdorf, "Mobile process description and execution," *Lecture Notes in Computer Science*, vol. 4025, p. 32, 2006.

[2] M. Wieland, D. Nicklas, and F. Leymann, "Managing technical processes using smart workflows," *Towards a Service-Based Internet*, pp. 287–298, 2008.

[3] J. Fleiss *et al.*, "Measuring nominal scale agreement among many raters," *Psychological Bulletin*, vol. 76, no. 5, pp. 378–382, 1971.

[4] C. Kunze, S. Zaplata, and W. Lamersdorf, "Mobile processes: Enhancing cooperation in distributed mobile environments," *Journal of Computers*, vol. 2, no. 1, pp. 1–11, 2007.

[5] S. B. Mokhtar, D. Fournier, N. Georgantas, and V. Issarny, "Context-Aware Service Composition in Pervasive Computing Environments," no. iii, pp. 129–144, 2006.

[6] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," in *Advances in Database Technology EDBT'98*, H. S. et al., Ed. Springer Berlin / Heidelberg, 1998, pp. 467–483.

[7] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[8] J. Herbst and D. Karagiannis, "Integrating machine learning and workflow management to support acquisition and adaptation of workflow models," in *Proceedings Ninth International Workshop on Database and Expert Systems Applications*, Vienna, Austria, 1998, pp. 745–752.

[9] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 7, pp. 215–249, Jul. 1998.

[10] W. M. van der Aalst et al., "Workflow mining: A survey of issues and approaches," *Data & Knowledge Engineering*, vol. 47, no. 2, pp. 237–267, 2003.

[11] T. van Kasteren, A. Noulas, G. Englebienne, and B. Kröse, "Accurate activity recognition in a home setting," in *Proceedings of the international conference on Ubiquitous computing*. ACM, 2008, pp. 1–9.

[12] G. M. Youngblood and D. J. Cook, "Data mining for hierarchical model creation," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 4, pp. 561–572, 2007.

[13] M. Schneider, "The semantic cookbook: sharing cooking experiences in the smart kitchen," in *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on*, Sep. 2007, pp. 416–423.