Fuzzy Event Assignment for Robust Context-aware Computing

Hannes Wolf, Jonas Palauro, Klaus Herrmann
Institute of Parallel and Distributed Systems, Universität Stuttgart, Germany
Email: {*hannes.wolf|klaus.herrmann*}@ipvs.uni-stuttgart.de, jonas@palauro.de

***Abstract*—User acceptance of context-aware applications relies on unobtrusive interaction and perceived dependability of the application. The accurate recognition and handling of high-level context information is a key factor , to achieve both. Currently, applications mostly work as isolated pieces of software and have to deal individually with the high uncertainties when recognizing and ambiguities when consuming high level context information. We use Adaptable Pervasive Flows (APF), to overcome these limitations and present our Fuzzy Event Assignment (FEvA) algorithm to resolve the ambiguities when assigning context information to the applications. Our simulation results show assignment accuracies between 83% to 97% and an improved performance when dealing with false positive, out-of-order and missed context information.**

I. Introduction

Context-aware applications provide users support for a broad range of activities in everyday life. But unobtrusive application, demand for little explicit user interaction and context information as the main source of input, driving the execution [1]. Throughout the paper we consider an application that automatically documents a nurses tasks in daily patient care without the necessity of explicit user interaction as an example for this kind of unobtrusive support. However, the nurse will only accept this kind application if it deals with the uncertainties of context recognition in a robust way.

Recognizing the actions and other high-level context information of the nurse from the environment involves reading data from (uncertain) sensors, processing the data and composing the context information from different sources. The sensor readings can be quantified by accuracy and precision, but the processing amplifies the degree of uncertainty. A context management system (CMS) [2], [3] provides the context information to the application via a query interface or in an event-based fashion. If the CMS also supports uncertainty handling, it further supplies the application with the degree of uncertainty for the requested information. The application decides if the received context information is consumed from the CMS so that it will no longer be available for other applications. This is necessary because the uncertain information could be interpreted in multiple ways, leading to inconsistent behavior of the informed applications. If the uncertainty is too high, the application discards the context. Similarly, the application could provide some policy, which allows the CMS to make those decision instead, but either approach leads to ambiguities when consumerist context.

We claim, that the *perceived dependability*, i.e. dependability from a user's point of view, of a context-aware application is conditioned by two factors: 1) the handling of uncertainties in context information and 2) the resolution of ambiguities when actually consuming context.

CMS provide sophisticated methods dealing with uncertainty of primary context, like location information [4] and for high-level context information – like needed to document the activities of the nurse – uncertain context reasoning or event correlation can be applied [5], [6]. However there is no system that takes the structural or contextual relation between the different applications into account to resolve the ambiguities, when assigning context information to the right application.

The algorithm we propose to robustly solve the assignment builds on Adaptable Pervasive Flows (APF) or flows for short. Flows originate from classical workflows, and were recently proposed as a programming paradigm for pervasive applications [7]. A flow basically consists of a number of activities $a \in A$ that with directed transitions $t \in T$, which define a partial execution order for the activities. An activity in a flow either represents some computational task, e.g. writing a database record or invoking a Web Service, or it specifies a task that a human has to perform in the real world, such as our nurse administering medicine to a patient.

Our newly developed *Fuzzy Event Assignment* (FEvA) algorithm supports the execution of flows, providing a robust yet flexible dynamic assignment of *context events* to single *activities* in the flow. FEvA determines all the activities, that could be interested in the available context information, based on the flows structural information and its current execution state. FEvA coordinates a competition between the activities, that is based on fuzzy logic, weighting the events and selecting the most appropriate candidates. Finally, the candidates are assigned, in such a way that a successful execution becomes more probable. We have implemented a simulation and tested our algorithm against false positive context information, context that occurs out-of-order, and missing context. The results show an avg. assignment accuracy of 91%.

The rest of this paper is structured as follows: In the next Section we introduce the assumptions we make in our system including the context model. After that, we present FEvA in Section III. Then, we show and discuss the results of our evaluation in Section IV. Finally we put our approach in perspective to related work in Section V and conclude the paper in Section VI.

 ISBN: 978-1-61208-149-6

II. Basic Assumptions and Models

First, we introduce some basic concepts of flow execution and then introduce our context model. Following that, we define the failure model on the context information the application has to deal with.

Applications that run in our system are flows. At development time, a programmer creates a *flow model* $\mathcal{F}$ of the application (documenting the daily routine of a nurse) that acts as a template. An *instance* of the flow is created at runtime, e.g., for a specific nurse, specific day, and executed on a flow engine. Most of the activities in this flow map to real-world activities of the nurse. Therefore we use a CMS that provides the flow engine with *context events*.

***Definition** 2.1 (Context Event):* A situation that can be recognized in the real world is referred to as event $e \in U$, where U denotes the universe of all events that the CMS can measure.

As the recognition relies on (uncertain) sensor readings, processing and composition of low-level context, an event is always uncertain. Currently, we assume that this uncertainty solely arises from the recognition process, i.e. the nurse in the real world always behaves correctly wrt the flow.

Events that represent semantically similar context can belong to a common *event type*, where each event belongs to at least one.

***Definition** 2.2 (Event Type):* An event type $E \subset U$ contains a number of individual events $E := \{e_1, \ldots, e_n\}$. A single event can be a member of different event types.

The purpose of an event type is twofold: First, it allows the programmer to simply select the most appropriate context the activity should respond to. A flow $\mathcal{F}$ defines a function $\epsilon : A \times \mathbb{N} \rightarrow \mathcal{P}(U)$ that maps activities to a number of distinct event types. $\mathcal{P}(U)$ denotes power set over the universe of events and $epsilon(a, i)$ yields the ith event type of activity a and $\emptyset$ otherwise. We write $\epsilon(a)$ for short, when referring to all event types of an activity. Second, the related semantics of the grouped events allow a more accurate recognition and classification. Events that are not contained in the expected event type are likely out of scope. The flow engine registers the event types $\epsilon(a)$ of a running activity at the CMS and receives an *event instance*, that indicates which event actually has been detected and also provides the degree of uncertainty.

***Definition** 2.3 (Event Instance):* Let E be an event type. An event instance $I_E : E \rightarrow [0, 1]$ defines a probability distribution function, where $\sum_{e \in E} I_E(e) = 1$.

We use probability theory, but the definition could be adjusted to incorporate other measures of uncertainty as well. This concludes the context model when considering single activities in the flow or single situations in the real-world. However, for successful flow execution we have to consider a number of event instances and their order, uncertainty and type. Therefore we define an *event sequence*.

***Definition** 2.4 (Event Sequence):* Let $\mathcal{E} := \{E_1, \ldots, E_j\}$ be the image set of ϵ for a given flow $\mathcal{F}$, i.e. $\mathcal{E}$ contains all event types used in $\mathcal{F}$. An Event Sequence $S := (I_E^1, \ldots, I_E^k)$ is an ordered list of k event instances, where $E \in \mathcal{E}$ can be of any type.

Given $\mathcal{F}$, we call S a valid sequence if it leads to a successful execution of $\mathcal{F}$. While event types are easily checked, we already showed in previous work how the uncertainty could be decreased using the flow structure [8]. However, as we motivated in the introduction, it remains a challenge to assign the event instances to the right activities. Therefore, ordering and actual occurrence of the events in the sequence are crucial. In the following, our we present our failure model covering three failure types.

The first failure type are false positives. The context system sometimes notifies the application of an event that did not occur in the real world. We define α as the percentage of added false positive events in a sequence S. For example, let S be a valid sequence and $\alpha = 0.05$, then the size of modified sequence S_α would be $|S_\alpha| \approx |S| * 1.05$. We assume that added event instances are uniformly distributed over the sequence, their type is randomly picked from $\mathcal{E}$ and the probability distribution of the instance is similar to the others in the sequence i.e. they can not be distinguished from the correct events in the sequence when inspecting the distribution.

Second, there are out-of-order events. Due to network transmission delay in a distributed CMS, temporary sensor failures or delay when running the situation detection operators on low-level context, the events in a valid sequence may be shifted. We define γ as the percentage of events that have not been affected by a sequence shift. For example, given $\gamma = 85$, 85% of the events have not been affected, but the remaining 15% are shifted in time (either way) according a normal distribution $\mathcal{N}(0, \omega)$. The ω is chosen such that 15% of the samples are larger than ± 1. The integer part of the sample indicates the shift and direction in the event sequence relative to its original position in the sequence.

Finally, there are events that happened in the real world, but the CMS simply missed them. This might be due to sensor unavailability, a bad reading, or the removal of a reading due to high uncertainty during the situation detection. Let δ denote the percentage of events from a valid event sequence that have been missed. So a value of $\delta = 10$ results in $|S_\delta| \approx |S| * 0.9 = |S| * (1 - \delta \backslash 100)$. The three failure models we presented can be combined and applied to a single event sequence, written as $S_{\alpha,\gamma,\delta}$.

III. Fuzzy Event Assignment

The goal of FEvA is to interpret a given a recognized event sequence $S_{\alpha,\gamma,\delta}$ for a flow $\mathcal{F}$, so that it is a valid sequence and leads to the same execution path as the original sequence S. FEvA tries to "find" the original events and map them to the right activities and, given enough evidence from existing events, also tolerate that some events are missing. We first introduce more details on *activity states*, the activity life cycle, flow execution semantics and our extensions to the activity state space. Then, we describe how FEvA fuzzifies incoming event instances and provides them as possible candidates for the competing activities using the *candidate selection* algorithm. As the execution of an activity progresses

 ISBN: 978-1-61208-149-6

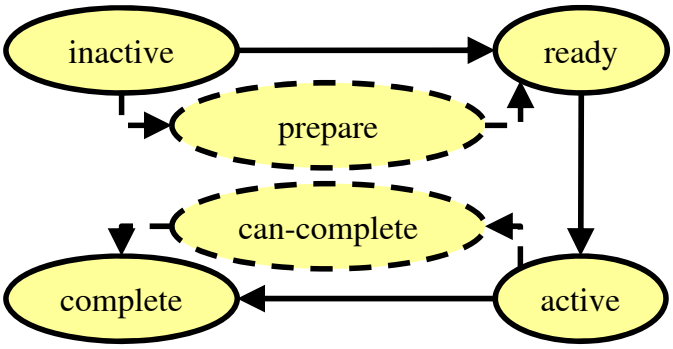


Fig. 1. Activity State Machine

it will eventually complete and the *event assignment* algorithm finalizes the event assignment and resolves possible conflicts.

A. Flow Model Semantics and Activity State Extension

As mentioned in the introduction, a flow basically consists of a directed acyclic graph $G = (A, T)$ with activities $a \in A$ as nodes and directed transitions between activities $t \in T \subset A \times A$ as edges. Each transition can be annotated with a logic condition, which depends on the received context information of the originating activity. Furthermore, some of the activities are mandatory, and must be completed successfully for a successful flow execution.

While being executed, an activity a assumes four distinct states that indicate its completion progress. These states are in order of execution $Z = \{inactive, ready, active, complete\}$. When an instance of $\mathcal{F}$ is created, all activities are in the inactive state. Meeting all prerequisites for execution, an activity a switches to the ready state. The flow engine then registers its event types at the CMS. Having received the first event, a assumes the active state. When a has received the last event, the conditions of the outgoing transitions are evaluated and a reaches the *complete* state. Following activities may be set to the ready state, depending on the condition evaluation results. The execution of the whole flow instance is considered successful if no activity is currently running i.e. in the active state and all activities that are mandatory for the successful execution are completed. The state machine for an activity is also depicted in Figure 1, considering only states and transitions with continuous lines. For more details on the formal flow model and the execution semantics refer to our previous work [8].

We extend the activity state space for FEvA with two additional states: $Z' := Z \cup \{prepare, can\text{-}complete\}$. Further, let $\omega : A \rightarrow Z'$ be the function that retrieves the current state of an activity a. If the state of the preceding activities a_x with $(a_x, a) \in T$ is $\omega(a_x) \geq ready$, then a switches from *inactive* to *prepare*. The event types of a prepared activity are also registered at the CMS. Therefore the number of registered event types increases and the chances of missing an out-of-order event are reduced. If an event is recognized out-of-order in current systems, the flow engine has not yet registered the event types at the CMS and the event is dropped.

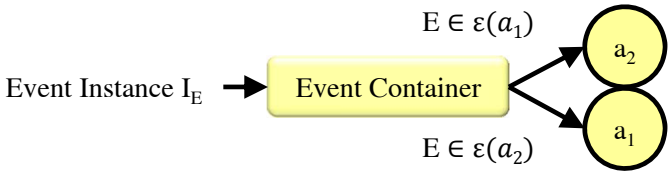


Fig. 2. Event Container Principle

Instead, the early events can now be cached by the flow engine until the prepared activity actually becomes *ready*. Before switching from *active* to *complete*, an activity first assumes the *can-complete* state. The state indicates that a has selected candidates for all its event types but the preceding activities have not yet reached the *complete* state. Waiting for their completion, we avoid that a consumes events that are possibly more suitable candidates for the predecessors while a better fitting event for a might still arrive. However, the conflict resolution mechanisms, which we will introduce later, will occasionally bypass this rule. The extensions of the state machine are also depicted in Figure 1 as states with dashed borders and dashed state transitions.

B. Fuzzy Event Assignment

As mentioned earlier FEvA, consists of two algorithms, one for event candidate selection and one for event assignment. Both algorithms are plugged into the *event container*, the component of the flow engine responsible for event caching and dispatching. Activities that have registered their event types at the CMS are known to the event container and stored separately for each event type as set of competing activities $C_E = \{a \in A | (\omega(a) \neq inactive \wedge \omega(a) \neq complete \wedge E \in \epsilon(a)\}$. The event container also stores a list of event instance candidates for each activity denoted as $candidates[a]$. Whenever the flow engine is notified about a new event instance I_E, it is stored in the event container.

The candidate selection algorithm, depicted in Algorithm 1, computes, which event instances are added to the list of candidates of an activity. First the algorithm computes a fuzzified representation of I_E. We utilize fuzzy sets (cf. [9]), each representing a linguistic value, defining the fitting quality of I_E for a single event from E. The individual fuzzy membership functions are defined as $\mu_x : [0,1] \rightarrow [0,1]$ where $x \in \{VL, L, M, H, VH\}$ is one of the linguistic variables "very low", "low", "medium", "high", "very high". The functions map the probability - or more generally the degree of uncertainty - of a single event given by I_E to a fuzzy membership value for the respective linguistic value. We used the same membership functions based on the standard triangular fuzzy functions for all combinations of activities and event types. For example, $e_1 \in I_E$ is the event where the nurse measures the pulse of the patient and $I_E(e_1) = 0.3$ then $\mu_M(I_E(e_1)) = 0.75$ and $\mu_H(I_E(e_1)) = 0.25$. As each event is weighted by every membership function, we further introduce the *fuzzy event weighting* function $\lambda : [0,1] \rightarrow [0,1]^5$ as concise version including all the

membership function results. Given $e_1 \in I_E$, $\lambda(I_E(e_1))$ yields $(\mu_{VL}(I_E(e_1)), \mu_L(I_E(e_1)), \mu_M(I_E(e_1)), \mu_H(I_E(e_1)), \mu_{VH}(I_E(e_1)))$, i.e. the mapping of the individual probability of the event to the fuzzified membership in all five fuzzy sets. The candidate selection algorithm computes the weighted event $\lambda(I_E(e))$ for all $e \in E$ and sends it to each activity, that is subscribed to E i.e. the activities in the set C_E. Each activity checks if any, and which, conditions have an at least "high" matching with I_E. They compute this matching using their own *fuzzy activity weighting* function $\kappa : [0,1]^5 \rightarrow [true, false]$, deciding if I_E is a suitable candidate event. We consider the mentioned example of pulse measuring $e_1 \in E$ again. An activity will chose I_E as possible candidate if and only if the lowest nonzero linguistic membership in $\lambda(I_E(e_1))$ is "high" or "very high" thus, $\mu_H((I_E(e_1) \geq 0.5 \vee \mu_{VH}(I_E(e_1)) \geq 0.0)$. Given this equation is fulfilled, the result of $\kappa(\lambda(I_E(e_1)))$ yields true and I_E is stored as a possible $candidate[a, E]$ for the activity a and the event type E. If I_E becomes a new candidate for an activity it further checks if it has the best overall fitting of the candidates available in $candidates[a, E]$. We denote the best fitting event instance as I_E^{max} where $\forall I_E \in candidates[a, E] : \mu_x(I_E^{max}(e)) \geq \mu_x(I_E(e))$ for the the highest non-zero linguistic membership value of I_E^{max}. Given that the new event instance $I_E = I_E^{max}(e)$ is the new best fitting one, the algorithm issues an assignment request for I_E that is later handled by the event assignment algorithm.

Algorithm 1 Candidate Selection Algorithm

```
Input: C_E, I_E
for e ∈ E do
  fuzzyWeights[e] ← λ(I_E(e))
end for
for a ∈ C_E do
  for e ∈ E do
    if κ(fuzzyWeights[e]) then
      candidates[a, E] ← candidates[a, E] ∪ {I_E}
    end if
  end for
  I_E^max ← max_{I_E}(candidates[a, E])
  issue_assignment_request(I_E^max)
end for
```

Having eventually received incoming events for all event types, an activity a changes its state to *can-complete*. But before it can commit its execution and reach the final *complete* state it must consume a single candidate event for each type from the event container. However, this might lead to conflicts, because a requested event instance I_E could also have been requested by another activity a_o. The event assignment algorithm is responsible for the final consumption of I_E and tries to resolve the conflicts. We omit a listing of the algorithm due to space reasons.

First the algorithm checks, if some $a_o \in C_E$ has also issued an assignment request. In case,this other activity has alternative candidates available , i.e $(candidates[a_o, E] \setminus I_E) \neq \emptyset$, we just select another candidate for a_o and assign I_E to a for consumption. When there is no other candidate available, we check if only one of the activities a or a_o is mandatory, preferring the mandatory one for the event assignment. Given that a now still lacks a non-requested candidate, we try to reevaluate the rejected candidates. If some of the other requested events for a have a fuzzy value that is very high, we relax the candidate criteria for E and check if more candidates become available. If this still does not yield a suitable candidate, the activity may be completed without having assigned an event to E. In order to do this a number of strict criteria have to be fulfilled. First, there is a fixed maximum number of allowed missing events per activity; in our case only one missing event is accepted. Second, the preceding activities of a must be *complete* and the succeeding activities must be in *can-complete*. Third the succeeding activities must have at least one event instance assigned with a very high fitting value. These rules ensure that there is enough evidence available, so we can assume that the necessary event has been missed and complete the activity nonetheless. However, most of the conflicts we mentioned will be resolved without using the conflict resolution mechanisms, when more of the correct events arrive. In the next section we assess the performance of FEvA wrt. the failure model we introduced in Section II.

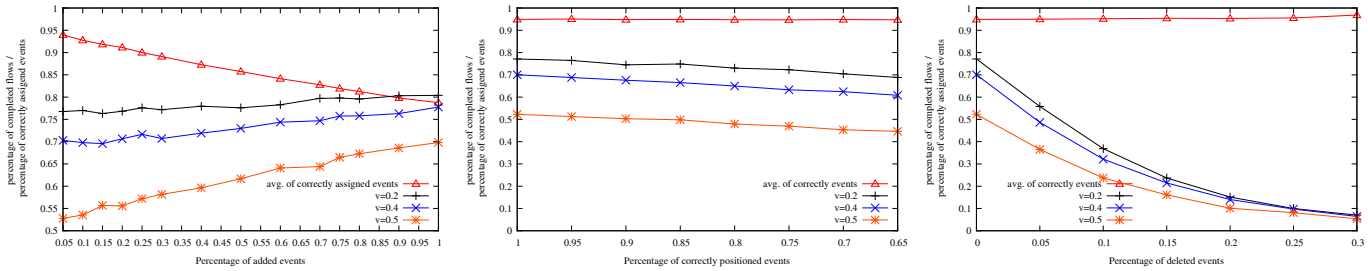
IV. Evaluation

To evaluate FEvA, we extended the existing simulation environment, developed in previous work [8]. In the following we present the setup of the individual experiments and the simulation parameters and then discuss the results.

A. Simulation Setup

In order to evaluate FEvA we need a suitable set of realistic flow models to test the algorithms in a wide range of cases. We created the flows from so-called workflow patterns. These patterns are common building blocks that have been identified in a number of real-world workflows, which include a high number of human tasks [10]. Furthermore, it has been shown that these patterns adhere to a co-occurrence distribution, indicating that some patterns follow others more regularly [11]. We used this co-occurrence distribution to generate the structure of the workflows used in our evaluations. The flows had sizes of 20, 30, 40 and 50 activities. For each size, we simulated 25 different flows and fed 100 different event sequences into each flow. Therefore every data point in the evaluation results is created from 10,000 flows executions.

The simulation is informed by two sets of parameters. The first set consists of the two values, *ground truth GT* and *variance V*, which are used to generate the probability distribution for the individual event instances. The GT is the probability that the CMS detects the correct situation that happened in the real world. The remaining probability $1 - GT$ is geometrically distributed to the other events of the event type. The V adds noise to the resulting distribution, i.e. the probability of each event of the pdf is altered by V and the final distribution again normalized. We simulated GT for values from 0.4 to 1.0 in steps of 0.1 and V from 0.05 to 1.0. To show



(a) added events α (b) moved events γ (c) deleted events δ

Fig. 3. Simulation Results - FEvA performance

the results clearly, the figures depicted only contain the graphs for $GT = 0.5$ which is FEvA's threshold for accepting an event (cf. Section III-B) and variances of $0.2, 0.4, 0.5$ representing a rather low, medium high and high amount of noise.

The second set defines the parameters α, γ and δ according to our definition in Section II. For α we chose values between 5% and 100%, for γ between 100% and 65% and for δ between 0% and 30%.

B. Results

The main results of the simulation for each of the parameters α, γ, and δ are depicted in Figure 3. The other values are set to their default value, i.e. 100 for γ and 0 for α and δ.

The results for the added false positives are the most surprising (c.f Figure 3(a)). While the number of correct events assigned to the activities decreases slowly from about 94% to 78%, the actual number of successfully completed flows increases. This is due to the fact, that the added false positives also take part in the assignment process and, especially in conflict situations, may be mapped to an activity too early and falsely, too. On the one hand this effect becomes stronger the higher the variance is; On the other hand the decrease in correctly assigned events also holds for the low variance but the effect cannot be identified right away. If we recognize the events with a high accuracy, the flow is able to deal with more false positives, but also assigns some of these false positives to the activities. However, for the worse context readings the result is counter-intuitive, as more flows complete because it is more likely that a fitting false positive exists.

When we confront FEvA with out-of-order events (c.f Figure 3(b)), the algorithm performs very well and tolerates the deviation. Most of the events are correctly assigned, i.e. well over 97% and the impact of the few falsely assigned events on the flow completion is low compared to the variance.

Considering the missed events (c.f Figure 3(c)), FEvA is still able to assign the remaining events accurately, again with well over 97%, but the missing events have a very strong impact on the flow execution. While a low number of missing events is somewhat tolerable, the amount of correctly completed flows drops rapidly to a mere 7% when more than a quarter of all events is missing. The difference between the graphs also becomes smaller. This shows that the deletion of events has a more severe impact on the correct execution than the variance. The mechanism we introduced to tolerate missed events somewhat helps, but there is a lot of room for improvement.

V. Related Work

We have investigated two different areas on work that is related to FEvA. We begin discussing the handling of fuzzy or uncertain (context) information in other workflow management systems, and continue with activity recognition systems, especially with a background in the health-care domain considering our example application.

The integration of context information into classic workflows used in enterprises has first been suggested by Wieland et al. [12] Their original approach does not consider uncertainty in context information, but in the meantime the authors provided a basic solution based on policies [13], which allows a workflow to specify a well defined behavior when dealing with uncertain context information and sensor failures. But their work actually lacks an algorithm, such as FEvA is, for matchmaking between uncertain context information and the workflow activities. Also from the area of business process management, Adam et al. [14], [15] proposed to use fuzzy logic to enable soft decisions in workflows based on the input provided to the workflow. However they did not consider uncertainties or ambiguities in the input information.

There are plenty of workflow models based on petri-nets (e.g. [16]), and also fuzzy petri net variants have been proposed [17] and applied to workflows [18]. Basically all elements of a petri net – places, markers and transitions – can be fuzzified and integrated into a fuzzy reasoning process. On the one hand, if we interpret an event instance as a fuzzified marker, our approach would be somewhat similar to the fuzzy petri nets. On the other hand the events represent external input, which has not been considered yet.

There have been numerous studies on activity recognition in the health-care domain. The major factors for decreasing the uncertainty and ambiguity in the recognition results are the selection of appropriate sensors, the available application model as well as the ease of sensor deployment and cost. For example, Barger et al. [19] studied a health status monitoring application and learned behavioral patterns of the user observing his daily activities using a number of motion sensors. But their system lacks an application model,

 ISBN: 978-1-61208-149-6

leading to missed events and false positives and a rather low recognition accuracy for uncommon situations. Najafi et al. [20] have built a monitoring system for elderly people using one acceleration sensor, and detecting position transitions and mode of locomotion. While performing very well for single transitions in a specific test scenario, the authors admit that for extended periods of time the sensing quality decreases. Finally, Biswas et al. [21] investigated different scenarios for elderly monitoring in home and professional-care scenarios. They used a complex and most likely expensive sensor setup for activity recognition, tailored for a specific scenario, state of the art recognition methods and very promising results. However their application model informing the recognition is basic and has been created manually. The authors specifically remark that knowledge from domain experts should be encoded in the recognition process. A flow is a very detailed representation of expert application knowledge, that we used to resolve the ambiguities when mapping the events to the activities.The presented approaches all use sophisticated activity recognition techniques, but do not consider the kind of application knowledge, that a flow could provide. In summary, the FEvA approach is a unique algorithm bridging the gap between activity recognition and context aware applications, dealing with ambiguities when consuming the recognized events.

VI. Conclusions and Future Work

FEvA, our new algorithm to resolve ambiguities when consuming uncertain context information, has demonstrated its effectiveness under the provided failure models. It achieves a reasonable assignment when facing a large number of false positive events and works very well when facing out-of-order events. We were able to limit the impact of a small number of missing events. We conclude that FEvA would be a very useful supplement for systems and environments where a lot of context information drives structured applications, such as the health-care documentation scenario we mentioned. In this scenario, FEvA significantly improves the perceived dependability of context-aware applications, advancing their user acceptance.

However, there is room for improving FEvA. Currently, we aim for a better mechanism to deal with the missed events. Furthermore, investigating the effects of different weighting functions per activity could lead to interesting results. Finally, it would be interesting to extend the approach not taking only one but multiple flow into account.

VII. Acknowledgments

The work described in this paper is partially funded by the 7th Framework EU-FET Project 213339 ALLOW

References

[1] Weiser, M.: The computer for the 21st century. Scientific American **265**(3) (September 1991) 94–104

[2] Kjaer, K.E.: A survey of context-aware middleware. In: SE'07: Proceedings of the 25th conference on IASTED International Multi-Conference, Anaheim, CA, USA, ACTA Press (aug 2007) 148–155

[3] Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing **2**(4) (2007) 263–277

[4] Lange, R., Weinschrott, H., Geiger, L., Blessing, A., Dürr, F., Rothermel, K., Schütze, H.: On a generic uncertainty model for position information. In Rothermel, K., Fritsch, D., Blochinger, W., Dürr, F., eds.: First Internationa Workshop on Quality of Context, QuaCon 2009. Number 5786 in LNCS, Stuttgart, Springer (June 2009) 76–87

[5] Koch, G.G., Koldehofe, B., Rothermel, K.: Cordies: expressive event correlation in distributed systems. In: Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems. DEBS '10, New York, NY, USA, ACM (2010) 26–37

[6] Choudhury, T., Philipose, M., Wyatt, D., Lester, J.: Towards activity databases: Using sensors and statistical models to summarize people's lives. IEEE Data Eng. Bull. **29**(1) (2006) 49–58

[7] Herrmann, K., Rothermel, K., Kortuem, G., Dulay, N.: Adaptable Pervasive Flows–An Emerging Technology for Pervasive Adaptation. In: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, IEEE Computer Society (2008) 108–113

[8] Wolf, H., Herrmann, K., Rothermel, K.: Robustness in Context-Aware mobile computing. In: IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'2010), Niagara Falls, Canada (10 2010)

[9] Zadeh, L.: Fuzzy sets. Information and Control **8**(3) (1965) 338–353

[10] Chiao, C., Iochpe, C., Thom, L.H., Reichert, M.: Verifying existence, completeness and sequences of semantic process patterns in real workflow processes. In: Proc. of the Simpsio Brasileiro de Sistemas de Informao. Rio de Janeiro: UNIRIO, Brazil (2008) p. 164–175.

[11] Lau, J.M., Iochpe, C., Thom, L.H., Reichert, M.: Discovery and analysis of activity pattern co-occurrences in business process models. In: ICEIS (3). (2009) 83–88

[12] Wieland, M., Kopp, O., Nicklas, D., Leymann, F.: Towards context-aware workflows. In Pernici, B., Gulla, J.A., eds.: CAiSE07 Proceedings of the Workshops and Doctoral Consortium. Volume 2., Trondheim Norway, Tapir Acasemic Press (Juni 2007)

[13] Wieland, M., Käppeler, U.P., Levi, P., Leymann, F., Nicklas, D.: Towards Integration of Uncertain Sensor Data into Context-aware Workflows. In in Informatics (LNI), G.E.L.N., ed.: Tagungsband INFORMATIK 2009 Im Focus das Leben, 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Lübeck, Lecture Notes in Informatics (LNI) (September 2009)

[14] Adam, O., Thomas, O., Martin, G.: Fuzzy WorkflowsEnhancing Workflow Management with Vagueness. In: EURO/INFORMS Istanbul 2003 Joint International Meeting. (2003) 6–10

[15] Adam, O., Thomas, O., Vanderhaeghen, D.: Fuzzy-set-based modeling of business process cases. In: ICCBR Workshops. (August 2005) 251–260

[16] van der Aalst, W.M., van Hee, K., Houben, G.: Modelling and analysing workflow using a petri-net based approach. In: Proc. 2nd Workshop on Computer-Supported Cooperative Work Petri nets and related formalisms. (1994) pp 31–50

[17] Pedrycz, W., Gomide, F.: A generalized fuzzy petri net model. Fuzzy Systems, IEEE Transactions on **2**(4) (November 1994) 295 –301

[18] Raposo, A., Coelho, A., Magalhaes, L., Ricarte, I.: Using fuzzy petri nets to coordinate collaborative activities. In: IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th. Volume 3. (2001) 1494 –1499 vol.3

[19] Barger, T., Brown, D., Alwan, M.: Health-status monitoring through analysis of behavioral patterns. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on **35**(1) (2005) 22 – 27

[20] Najafi, B., Aminian, K., Paraschiv-Ionescu, A., Loew, F., Bula, C., Robert, P.: Ambulatory system for human motion analysis using a kinematic sensor: monitoring of daily physical activity in the elderly. Biomedical Engineering, IEEE Transactions on **50**(6) (2003) 711 –723

[21] Biswas, J., Tolstikov, A., Jayachandran, M., Fook, V.F.S., Wai, A.A.P., Phua, C., Huang, W., Shue, L., Gopalakrishnan, K., Lee, J.E.: Health and wellness monitoring through wearable and ambient sensors: exemplars from home-based care of elderly with mild dementia. Annales des Télécommunications **65**(9-10) (2010) 505–521

 ISBN: 978-1-61208-149-6