FlexCon – Robust Context Handling in Human-oriented Pervasive Flows

Hannes Wolf, Klaus Herrmann, and Kurt Rothermel

Institute of Parallel and Distributed Systems,
Universitätsstraße 38, D-70569 Stuttgart, Germany
{hannes.wolf|klaus.herrmann|kurt.rothermel}@ipvs.uni-stuttgart.de

Abstract. Workflows are increasingly becoming a universal means for driving and coordinating complex processes, not only in the business world but also in areas like pervasive computing. *Pervasive flows* run in parallel with the user's real-world actions and are synchronized using automatically collected context information about her current activities (context events). Respective workflows cannot be rigidly defined since the user needs to retain her flexibility and must not be obstructed by the workflow. However, if the order of activities is not defined until the activities are actually executed, correctly assigning the uncertain context events becomes a major challenge. We propose FlexCon – a novel event assignment approach for such human-oriented workflows that is based on hybrid workflow models and Dynamic Bayesian Networks. FlexCon exploits the dependency between context events to provide more accurate information as to which events need to be consumed by which workflow activities. Our evaluations show that FlexCon improves the event accuracy on average by 54% and the number of successful completed flows on average by 88%. Thus, FlexCon represents a major step towards unobtrusive pervasive applications.

1 Introduction

Workflows are an adequate means for modeling the functionality and the temporal flow of complex activities in many areas of information technology. Traditionally, they are rooted in business applications [1], where processes span diverse departments of the same company or different companies. In recent years, however, an influx of workflow technologies into the domain of pervasive systems has started. They have been proposed as useful tool for environments with intensive human interaction [2]. Gradually, workflows have become more flexible, supporting [3] and also gained context-awareness [4].

In the ALLOW project [5], we investigated the concept of *Adaptable Pervasive Flows* (in short *flows*) as a means for rendering complex applications and the environment (technical equipment, information systems etc.) adaptive to the mobile user. The basic idea of flows is that a process that a user has to execute is modeled as a flow. This flow is running in the background in parallel to the actual real-world actions of the user and can support her in different ways. The key to this is that the flow requires as little explicit input from the user as possible in order to offer unobtrusive support. First, the flow can guide the user through the process by giving her feedback in critical situations (e.g. visual or audio). Second, it may take over certain routine tasks automatically (e.g.

documenting the actual process for legal or quality-related reasons). Third, the flow may prepare the environment (e.g. configuring electronic devices etc.) in advance to minimize disturbances and the work load of the user. Fourth, the flow may automatically adapt itself in case it recognizes that the planned execution will fail (e.g. due to lack of a required resource). All of this is possible since the flow "knows" the prospective future activities associated with the process – they are part of the flow model.

To achieve this with minimal explicit input, the flow system monitors what the user does in the real world in order to automatically synchronize with her actions and drive the flow forward respectively. This monitoring, which provides the majority of the input to the flow, is done by means of *activity recognition* [6, 7] using different types of sensors. The respective data is provided to the flow as so-called *context events*.

As an example, consider a nurse in a hospital that has a flow modeling the morning routine of a patient. That flow involves different activities like `washing`, `dressing`, `measuring blood pressure`, and `disinfecting hands`. The actions associated with these activities in the real world are detected by the activity recognition system and provided to the flow as context events. Some of the activities have a clear ordering relation, e.g. she has to wash the patient *before* she dresses him. Others are mandatory but may occur at different stages of the flow (e.g. `measuring blood pressure`). Yet other activities like `disinfecting hands` may be carried out multiple times as needed. Thus, a respective flow must not be defined rigidly with a fixed predefined order between activities. It must allow the nurse the flexibility of executing the process in her own way and in the way required by the specific situation.

When the flow arrives at a certain activity, there may be different ways of continuing (different possible next activities). Depending on the context events received, the flow has to decide which of these paths the user took. Supplying the correct context events to the right flow activities is a tough challenge under these conditions. First, events may be noisy, duplicated, deleted due to failure or delayed. This is a fundamental problem that occurs irrespectively of the flexibility in the flow models [8]. Second, due to the gained flexibility, it may not be completely clear which activity is actually executed next and thus awaits a respective event.

To solve this fundamental problem of pervasive flow technology in general, we propose FlexCon, a system that leverages a combination of imperative (rigid) and declarative (flexible) flow models, Dynamic Bayesian Networks (DBN), and particle filters to reduce the uncertainty of context events. FlexCon builds probabilistic models of the dependencies between events and uses this information to improve the processing of probabilistic context events. We evaluated FlexCon based on a real-world hospital study and found that it decreases the context event uncertainty by up to 73%. While standard flow technology exhibits a high flow failure rate[1] of 82% under the conditions explained above, the failure rate of FlexCon is only 65% on average of all flows. This presents a major step forward in the areas of workflow-based pervasive computing.

The rest of the paper is structured as follows: In Section 2 we will investigate the related work in relevant areas. We present the basic models of context and workflows in Section 4. After that we introduce our FlexCon approach in Section 5 and evaluate

[1] A flow fails if context events are assigned to false activities or are discarded due to false recognition results.

it based on a real-world scenario in Section 6. Finally, we present our conclusions in Section 7.

2 Related Work

In the following, we will investigate the state-of-the-art in the relevant areas of research. We will first discuss activity recognition systems and how they deal with context uncertainties. While our work is not directly associated with this area, it does provide a new approach for handling the uncertainties perceived in activity recognition systems on a higher layer by exploiting application knowledge. Subsequently, we will take a closer look at the field of context-aware workflows.

There have been numerous studies on activity recognition in the health-care domain [9–11]. The major factors for decreasing the uncertainty in the recognition results are the selection of appropriate sensors and exploiting available application models. Biswas et al. [11] specifically remark that the recognition process can benefit from the knowledge of domain experts. A flow is a very detailed representation of expert application knowledge, that FlexCon uses to increase the accuracy of events.

Barger et al. [9] studied a health status monitoring application that learns behavioral patterns of a user from his daily activities using a number of motion sensors. But their system lacks an application model too, leading to missed events and false positives and a rather low recognition accuracy for uncommon situations.

Najafi et al. [10] have built a monitoring system for elderly people using one acceleration sensor, and detecting position transitions and mode of locomotion. While this approach performs very well for single transitions in a specific test scenario, the sensing quality decreases over extended periods of time due to the lack of an application model.

The presented approaches all use sophisticated activity recognition techniques, but do not consider the kind of application knowledge a flow provides, thus, neglecting the huge potential.

The integration of context information into classic workflows used in enterprises has first been suggested by Wieland et al. [4], who provide interfaces for accessing context information from within a workflow. This approach was later extended to deal with Quality of Context [12]. Here, a policy language is used to define the acceptable amount of uncertainty in context information and to filter out information that does not match the specified criteria. This approach is based on the idea to simply prevent the workflow from receiving uncertain information. However, if a workflow does not receive the information at all, this can be just as detrimental as receiving false information. We go one important step further by improving the information such that it becomes useful for the flow.

Adam et al. [13] proposed 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. Their approach aims at making a better decisions from the business perspective.

PerFlows, presented by Urbanski et al. [14], are context-aware workflows that are suitable for pervasive scenarios and provide flexible activity scheduling and processing. However, they require heavy user interaction to work properly. This is disadvantageous in scenarios where the workflows shall run in the background in order not to obstruct the user. In our own previous work [15], we presented an approach for dynamic context-awareness suited for pervasive flow-based applications. But this approach also neglects the handling of uncertain context information.

In 2010, we have proposed FlowCon [15], the first system that was able to decrease the uncertainty of events by learning the dependencies between events and by explicitly exploiting the temporal structure encoded in imperative flows (flows with a strict temporal ordering among activities). FlowCon can increase the number of successfully executed flows by a factor of 6 to 8 under normal conditions. With the FlexCon system presented in this paper, we build on this work and apply related technologies to hybrid flow models that are more flexible and allow users to act more freely. Overall, this represents a significant step forward in this field.

3 Scenario

The application scenario we use to evaluate FlexCon is from the health care domain. We studied the processes conducted by nurses in a geriatric ward in Mainkofen, Germany over a period of 14 days.

Through a mining process, we extracted workflows from the observations made at the ward. The respective processes have not been defined as workflows before. However, in this highly structured working environment, workflows are implicitly followed in order to fulfill a number of standards in terms of patient care. In total we collected 32 datasets from 15 different nurses, where each dataset covers the care of 3 to 5 patients, yielding a total of 130 observed workflow executions.

The purpose of applying a system of adaptable pervasive flows in this institution is twofold: First, the activities shall be automatically documented for the records for quality control, process improvement, and legal reasons. Second, the flow system shall give guidance in case the standard procedures are not followed in order to avoid mistakes and help inexperienced personnel in learning the procedures.

A typical workflow, e.g. from the morning routine, consists of 30 to 50 activities of which about 20% have no strict order. The entire navigation in such a flow depends on context events (i.e. the correct next activity is chosen based on the context events received). An example fragment is depicted in Figure 1. Solid boxes depict mandatory *activities* that need to be executed unconditionally while dashed boxes are optional. For example, a_2 and a_3 are optional activities. The execution of a flow instance is valid if, one of the optional activities, both or non of them have been executed, while a_1, a_4, and a_6 need to be executed for the flow to be successful. Solid arrows are *transitions* that imply a strict ordering between the activities: a_1 must be followed by either a_2 or a_3 and a_4 must follow both a_1 and a_2. The dashed lines are *constraints* that define certain restrictions on the execution order of the related activities. The figure depicts two examples: the semantics of the *not succession* constraint between a_3 and a_5 is that a valid flow execution must not contain both activities. It may contain either one or none of them, and if one is executed, it can be executed arbitrarily often. As we will explain in Section 4, constraints can be arbitrary linear temporal logic expressions. Some of them have been translated into a graphical representation.

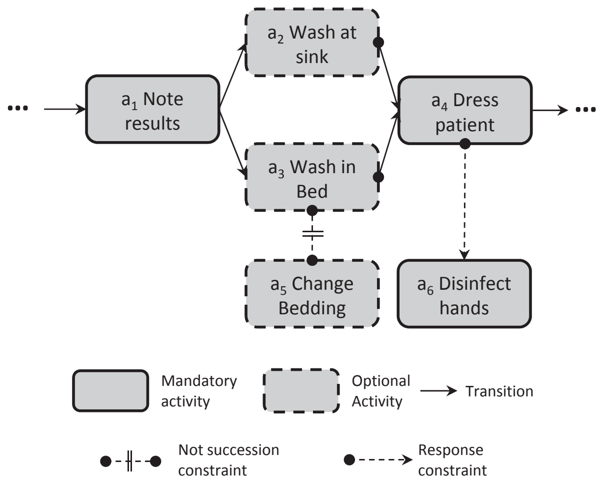


Fig. 1. Example workflow from a hospital scenario

The flow shown in Figure 1 is a fragment of a larger flow that models the actual processes found in the Mainkofen nursing ward. Its overall semantics is the following: When a nurse arrives at this fragment, she must document the results (a_1) of the preceding steps, which include some regular morning examinations, such as measuring blood pressure. As these examinations are carried out without assistance of an electronic device, the flow ensures that the nurse will not forget the results during the following steps. Then she has to take a decision: she may wash the patient at the sink (a_2) or in his bed (a_3), depending on the patients condition and mood. In FlexCon actually both activities are entered as soon as a_1 has completed. Depending on the incoming context events, either one or both are executed. If the nurse decides to wash the patient in his bed (a_3), she cannot change the bedding (a_5) since the patient still never leaves his bed during the whole procedure (this is done in a different flow). After the nurse has completed the washing activity, she needs to dress the patient (a_4). When she dressed him, she must disinfect her hands (a_6) at some later point in time, possibly after a number of other intermediate activities. But, she may disinfect her hands at any point in time, while the flow is being executed. This is beneficial in two ways. First, the nurse can flexibly decide to disinfect her hands multiple times, e.g. during washing the patient, also allowing the system to keep track of her personal hygiene as well as the patients. Second, the flow can guide the nurse to disinfect her hands before she continues to care for another patient, this way enforcing the hospitals hygiene rules.

4 Flow and Context Models

In this section, we will first define our model of context information before we give a definition of hybrid flows.

4.1 Context Model

As the flows should not obstruct the nurse in her daily routine, they are solely driven by context events. Therefore, adequate sensors and an activity recognition and context management system (CMS), must be available to gather context information and provide the context events. However, state of the art activity recognition systems have some drawbacks. Either, they require the precise deployment of (expensive) sensors, or the setup and training of the system is tedious. Cheaper activity recognition systems, e.g. based on standard smart phones, only provide moderate recognition rates, at best. While the former technology might be applicable in high-cost environments such as an operating room, we have to rely on the latter kind in the area of cost-sensitive every-day patient care.

In the scope of our scenario, we assume that in practice the type of different events FlexCon is interested in, is a finite set.

Definition 1 (Event Type). *A type of situation that can be recognized in the real world is referred to as an* event type $u \in U$*, where* U *denotes the universe of all event types that the CMS can measure.*

An event type describes the abstract semantics of an context event. For example, *nurse walking* could be an event type. Events of this type are created whenever a nurse changes her mode of locomotion to *walking*. Event types that represent semantically similar context can belong to a common *event type set*, and each event type belongs to at least one event type set.

Definition 2 (Event Type Set). *An event type set* $E \subset U$ *contains a number of event types* $E := \{u_1, \ldots, u_m\}, m > 0$*. A single event type can be a member of different event type sets.*

The event type set containing all event types for a nurse's locomotion modes could be, e.g. {*nurse walking*, *nurse sitting*, *nurse standing*}. The purpose of an event type set is twofold: First, it allows the flow modeler to simply select the most appropriate context the activity should respond to. As seen below, a flow model defines a function that maps every activity to a number of distinct event type sets. Second, the related semantics of all event types in an event type set allows for a more accurate recognition: Event types that are not contained in one of the expected event type sets of the current flow activity are likely to be out of scope. When executed the flow registers the event type sets of a running activity at the CMS and receives *event instances*.

Definition 3 (Event Instance). *An event instance* $e \in U_e$ *is an instance of a specific event type* $u \in U$*.* U_e *is the universe of all event instances occurring in the system.* e *belongs to a specific event type* $u \in E$*, and the uncertainty about which exact event type in* E e *belongs to is given by a probability distribution* $I^e_E : E \mapsto [0, 1]$*, where* $\sum_{u \in E} I^e_E(u) = 1$*.*

I^e_E is our basic model of uncertainty. Instead of saying that an event instance is of type u, the CMS provides the distribution I^e_E, and $I^e_E(u)$ is the probability that e is of type $u \in E$. For example if $u = nursewalking$ and $u \in E$ then $I^e_E(nursewalking) = 0.52$ indicates that the probability of e being of type *nurse walking* is 52%.

4.2 Hybrid Flow Model

A *flow model* is a template for a specific type of flow. A runnable instance of such a model must be created whenever a flow is to be executed. We call this a *flow instance*. In the following, we also refer to such an instance simply as a *flow*. The flow instance is executed by a flow engine. In our work we employ *hybrid flow models* that contain *transitions* as well as *constraints* between activities and, thus, are a mixture of classical imperative production workflows [1] and declarative flexible [16] workflows. Transitions are annotated with boolean *conditions* over the possible set of context events while *constraints* consist of *linear temporal logic* expressions that describe the acceptable temporal relation of two or more tasks (e.g. a must be executed before b). If a flow modeler currently wants to use a mixture of both modeling paradigms he is required to add this flexibility in a hierarchical way [17]. He must decompose the application into a number of hierarchical layers, usually representing a different level of abstraction and choose the best modeling paradigm for each layer. Our hybrid flow model, allows the use of both paradigms directly on all abstraction levels and can also be applied to applications where the hierarchical decomposition is not possible or introduces further complexity.

Definition 4 (Hybrid Flow Model). *A hybrid flow model $\mathcal{F}$ is a 4-tuple $\mathcal{F} := (A, T, C, L)$, consisting of a set of activities A, a set of transitions T, a set of conditions C, and a set of constraints L.*

Definition 5 (Activity). *An activity a represents an atomic piece of work within a flow. This includes invoking web services, internal computations, notifying a human about a task, or receiving context events indicating changes in the real world.. The set $A := \{a_1, \ldots, a_n\}$ defines all activities of a flow. An arbitrary number of event types can be added to each activity. Let $\epsilon_a : \mathbb{N} \mapsto \mathcal{P}(U)$ be the event type assignment function for a, where $\mathcal{P}(U)$ denotes the powerset over the universe of events types. Further, let k be the number of event types associated with a, then $\epsilon_a(i)$ yields the i-th event type for $i \leq k$, and $\emptyset$ for $i > k$. We write ϵ_a for short when referring to the set of all event type sets assigned to a. Furthermore activities may be marked as mandatory.*

Activities in a flow may be executed arbitrarily often and in any order, such as a_6. A flow can successfully complete its execution when all mandatory activities have been executed at least once. Transitions and constraints limit this flexibility and impose structural ordering on the flow activities, such as the response constraint between a_4 and a_5. When an activity is started it registers at the CMS for context events of its event types ϵ_a. As example, let $e \in U_e$ be an event instance of type u that the activity a_1 `note results` in the flow requires to complete its execution. Let $u = write$ and $u \in E_\alpha$, where E_α contains the event types {*wash, dry, write, fetch, disinfect*} representing some activities of the nurse. When the engine enters the execution of a_1, E_α is registered at the CMS.

Definition 6 (Transition). *Given a set of activities A, the set of all transitions within a flow is $T \subseteq A \times A$. A transition $t = (a_x, a_y)$ represents a directed control flow dependency from a_x to a_y with $a_x, a_y \in A$. A transition is annotated with exactly one transition* *condition, that is referred to as $c(t)$. Further, we define $d_{in}(a_i) \coloneqq |\{(a_x, a_y) \in T | a_i = a_y\}|$ and $d_{out}(a_i) \coloneqq |\{(a_x, a_y) \in T | a_i = a_x\}|$ as degree of incoming and outgoing transitions for an activity.*

The transitions allow certain control flow variants (cf. Figure 1): linear sequences ($d_{out}(a_4) = 1$), parallel branching ($d_{out}(a_1) > 1$) and joins like for ($d_{in}(a_4) > 1$), and combinations of those. Conditional decisions can be made taking the transition conditions into account.

Definition 7 (Context Condition). *A context condition c is inductively defined as $c \rightarrow u | (c_1 \vee c_2) | (c_1 \wedge c_2) | \neg(c_1)$ with $u \in U$ and c_1, c_2 are already valid conditions and the common semantics for the probabilistic logical operators.*

The condition $c(t)$ for $t = (a_x, a_y)$ is evaluated when a_x has received an event instance e for every ϵ_a. We insert the received event instances and check $c[u/I_E^e(u)] \geq t_n$ against the navigation threshold t_n. If the equation is fulfilled, the condition evaluates to true and the activity a_y is executed.

Definition 8 (Constraint). *A constraint l is an expression in linear temporal logic (LTL) that defines the temporal ordering of one or more activities in the flow. l is inductively defined as $l \rightarrow a | (l_1 \vee l_2)$(logical or)$|(l_1 \wedge l_2)$ (logical and)$|\neg(l_1)$ (logical negation)$|(l_1 \rightarrow l_2)$ (logical implication)$|\diamond(l_1)$ (eventually)$|\,\Box(l_1)$(globally)$|\, l_1 U l_2$(strong until), where $a \in A$ and l_1, l_2 are already valid constraints. The literals given in the expression l denote the completion of the respective activity a in the flow.*

Constraints can be grouped in different classes of constraints such as existence, (negative) relation, (negative) order [16] and provided in a graphical representation (c.f. Figure 1). At runtime they are converted to finite state machines (FSM) [18] and can be checked online for violations. If the FSM is in an accepting state the constraint is *valid*. When the FSM is not in an accepting state the constraint is *temporary violated*. The subsequent execution of further activities can eventually lead to a valid constraint. For example consider the response constraint for a_4 and a_6. It is *valid* as long as a_4 has never been executed. After a_4 has been executed, the constraint becomes *temporary violated* until a_6 has been executed. A constraint is permanently violated if the FSM reaches an error state and no sequence of activities can fulfill the constraint anymore. The response constraint can never be permanently violated. In contrast, the not succession constraint becomes *permanently violated* if both a_3 and a_5 have both been executed in the same flow instance. A flow can successfully complete its execution iff all constraints are valid. Arbitrary constraints are possible, but the common constraints are given in a graphical notation for the ease of modeling. For example, the not-succession constraint depicted in Figure 1 would be defined as $\Box(a_3 \rightarrow \neg(\diamond(a_5)))$.

The execution of the flow model yields a flow trace. When an activity is completed, this is recorded in the flow trace along with the event instances it received.

Definition 9 (Flow Trace). *A flow trace $\mathcal{T}$ is a sequence of completed activities $\mathcal{T} \coloneqq (a_1, \ldots, a_j)$ in ascending order of completion times. The event instances each activity has received are also stored within the trace. Let $\theta(\mathcal{T}, a, u) \mapsto e$ be a function that yields the event instance $e \in u$ associated with activity a in trace $\mathcal{T}$.*

From a single trace, it is possible to reconstruct the actual execution of a flow instance and which context information, i.e. event instances, lead to this execution. All traces are stored in a flow history documenting the executions for later analysis. We use the flow history of a flow model as the data set for training the FlexCon algorithm later.

5 FlexCon

We will first provide an overview of the working principle and architecture of FlexCon using a concrete example based on the scenario and flow we presented. Following that, we explain our method to create the DBN from the flow in detail and how we adopted particle filtering techniques for our approach.

5.1 Overview

The main goal of FlexCon is to decrease the uncertainty of an event instance e. This means, if e is of type u, then FlexCon shall collect additional evidence for this fact and increase the probability $p = I_E^e(u)$ for the event type u in the given distribution. To achieve this we use the flow as additional source of information. The flow model provides information concerning the structure (activities, transitions, constraints) of the flow and, thus, about the expected temporal relation of respective context events. The flow instance provides information given by its execution state, i.e. the current state of the activities and the already received context events.

Let us assume that the flow engine has started the execution of a_1, and receives the event of the types associated with a_1, including E_α (c.f. Section 4.2). In a system without FlexCon, the flow engine would simply compare the probability $p = I_{E_\alpha}^e(u)$ with the engine's *navigation threshold* t_n and execute the respective transition if $p > t_n$. This simple approach is depicted in Figure 2 on the left. FlexCon, in the other hand, uses the information encoded in the flow model and the flow instance to infer additional evidence for the fact that e is of type u. Thus, it improves the probability distribution $I_{E_\alpha}^e$ that is the basis for the threshold comparison, leading to a more robust flow navigation.

FlexCon uses *Dynamic Bayesian Networks* to interpret context events depending on the current state of the flow. A DBN is a probabilistic data structure that is flexible enough to represent the current flow state, the already received events, and the relation between the events according to the transitions and constraints of the flow model. FlexCon builds the structure of the DBN from the flow model and trains the DBN using traces of previously executed flows. This is shown in Figure 2 on the lower right. We explain the details of the construction algorithm in the next section.

When a flow instance is executed, every incoming context event e is sent to the DBN. Any such event is associated with a probability distribution I_E^e (cf. Definition 3). The DBN infers an *additional* conditional probability distribution $I_E'^e$ for e over E. The distribution I_E^e given by the CMS and $I_E'^e$ given by the DBN are combined, yielding an overall distribution $I_{E_\alpha}''^e$ which is then used by the flow engine to make its navigation decision. Our evaluations show that if $e \in u$ then, on average, $I_E''^e(u) > I_E^e(u)$. Hence, FlexCon reduces the uncertainty contained in the original distribution such that the flow engine can make more correct threshold decisions.

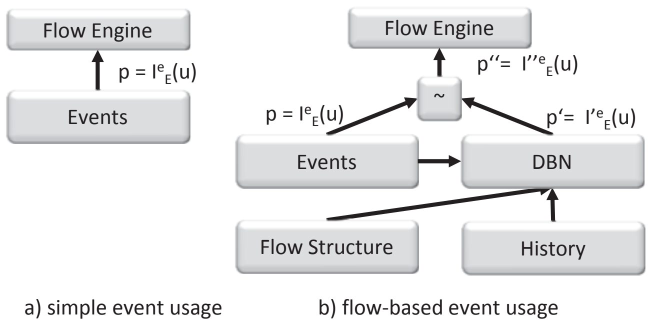


Fig. 2. Architecture overview

Using exact inference to get I'^{e}_{E} from a complex DBN, such as the one built from the flow model, is computationally infeasible. Therefore, we use an approach based on *particle filters* [19] to increase the performance. We adapted the standard particle filter approach to reduce the computational effort, which allows us to use more particles on a more sparse DBN network and achieve more accurate inference results. We present a detailed description of the inference algorithm in Section 5.3.

5.2 Dynamic Bayesian Network - Structure and Learning

A Bayesian Network $\mathcal{BN} = (\bar{X}, D)$ is a directed acyclic graph representing a joint probability distribution over a number of random variables (RVs) $\{X_1, ...X_n\} = \bar{X}$. $\bar{X}$ represents the nodes and the edges $D \subseteq \bar{X} \times \bar{X}$ define a conditional dependency from the source RV to the target RV. In FlowCon, we used BNs as the flows where based on imperative models that specify the complete execution order. Therefore, the simple static BNs were sufficient. The hybrid model in FlexCon, however, introduces much more freedom for the users to drive the flow forward in different ways and, thus, more dynamics. The static BN model does not support such a dynamically changing probabilistic process. Therefore, FlexCon employs Dynamics Bayesian Networks which are tailored for dynamically changing systems.

In a DBN [19, 20], the state of the RV changes over time and the observed values for the RV in the current *time slice* $\bar{X}_t$ depend on the observations of one or more previous time slices. This dependency is expressed by the *transition model* $\mathcal{TM} = P(\bar{X}_t|\bar{X}_{t-1})$. When we write $X_{1,0}$, we refer to the RV X_1 in the time slice $t = 0$. Additionally, a DBN has a prior distribution $\mathcal{PD} = P(\bar{X}_0)$ for time $t = 0$, such that the definition of a DBN is given as follows: $\mathcal{DBN} = (\bar{X}, \mathcal{TM}, \mathcal{PD})$[2].

DBN Construction Let $\mathcal{F}_1 = (A, T, C, L)$ be the flow model from our example in Section 3. For each $a \in A$ and each $E \in \epsilon_a$, FlexCon creates a node in the DBN. More formally, the function $\chi : A \times \mathcal{P}(U) \to \bar{X}$ maps an activity a and an event type set E

[2] Since FlexCon has no hidden variables, there is no need for a sensor model as it is usually found in the DBN definition

to a unique RV X of the DBN. Let further $\bar{\chi}(a, \epsilon_a)$ be the set of all RVs associated with activity a. $\chi(a, E) = X$ with $E \in \epsilon_a$ is discrete and can assume the same values present in the event type set E plus a *null* class, represented by $\perp$. For example, let us consider a_1 and $E_\alpha \in \epsilon_{a_1}$ (c.f. Section 4.2). The respective random variable $\chi(a_1, E_\alpha) = X_\alpha$ can assume any value from $\{$*wash, dry, write, fetch, disinfect*$, \perp\}$. $\chi(a, E)_t$ and $\bar{\chi}(a, \epsilon_a)_t$ refer to the respective RVs in time slice t.

The *time slices* in our DBN are defined with respect to the execution state of the flow: Every time an activity completes its execution and the flow state is changed accordingly, we enter the next time slice in the DBN. FlexCon creates the transition model (the time dependencies) from the transitions and constraints in the flow model. Both of them enforce an execution order on the set of activities. We map these order relations to the transition model, introducing directed edges (dependencies) from one time slice to the next. The strength of these dependencies in learned from flow traces (past flow executions) in a subsequent step. In the following, we describe the construction and learning phases first for transitions and then for constraints.

A transition $t = (a_x, a_y) \in T$ between two activities represents a very strong dependency as a_y can only be executed when a_x has been completed. Therefore, we create a dependency in the network for a pair of RVs if a transition exists between the respective activities as follows.

$$(\chi(a_x, E_x)_t, \chi(a_y, E_y)_{t+1}) \in P(\bar{X}_{t+1}|\bar{X}_t) \iff ((a_x, a_y) \in T) \wedge (E_x \in \epsilon_{a_x}) \wedge (E_y \in \epsilon_{a_y}).$$

For example, consider the activities a_1 and a_2 in Figure 1: They have a transition and, therefore, each $X \in \bar{\chi}(a_2, \epsilon_{a_2})_{t+1}$ would have $\chi(a_1, E_\alpha)_t$ as parent node, because $E_\alpha \in \epsilon_{a_1}$.

As constraints usually provide a less strict ordering of activities it is more difficult to derive the correct dependencies for the transition model. These dependencies can be different for each execution trace of the same flow. Let $l_1 = \Box(a_3 \rightarrow \neg(\diamond(a_5)))$ represent the *not-succession* constraint in the example in Figure 1. First, FlexCon assumes that there is a bidirectional dependency between all the activities that are contained as literals in the expression (a_3 and a_5 in the example). Hence, FlexCon adds $(X_{3,t}, X_{5,t+1})$ and $(X_{5,t}, X_{3,t+1})$, with $X_3 \in \bar{\chi}(a_3, \epsilon_{a_3})$ and $X_5 \in \bar{\chi}(a_5, \epsilon_{a_5})$ as dependencies in the DBN. In a second step, FlexCon determines the type of dependency that has to be included in the transition model $\mathcal{TM}$. If the sequential execution of the originating activity a_3 and the the target activity a_5 of the dependency *permanently violates* the constraint (as is the case in the example), FlexCon marks this dependency as *negative*. Negative dependencies are handled differently in the learning process as described below. If the sequential execution leads to a *valid* or *temporarily violated* constraint (c.f. Section 4.2), the dependency is handled like a transition. If the subsequent execution of the two activities has no influence on the constraint, we do not add a dependency at all. The latter is the case for the *response* constraint between a_4 and a_6 in Figure 1, where the execution of a_6 has absolutely no dependency on the execution of a_4.

DBN Learning In order to learn the strength of dependencies in the DBN, we use the flow history as training data, counting the occurrences of all event pairs and learning their joint probability distribution. The portion of the flow history that is relevant for the learning is controlled by a sliding window algorithm taking only a number of recent traces into account. This helps in controlling the effectiveness of the learning procedure in the face of a changing behavior of the flow system.

For dependencies originating from flow transitions, the simple counting algorithm as explained above is sufficient. For constraints, we have to apply a different mechanism: In order to learn the strength of negative relations, we increase the count of the *null*-class for every trace where no such event sequence could be observed. This leads to a reduced probability of any other event type of the respective event type set. As an example, consider the *not succession* constraint of a_3 and a_5 again. The execution of a_3 will indicate that a_5 is never going to happen in any valid execution of this flow instance. Therefore, we reduce the belief of the DBN that any of the events associated with a_5 is likely to be recognized. An inexperienced nurse may execute the activity sequence a_3, a_5 nonetheless, but the flow can provide guidance for this case, preventing the nurse from violating the constraint l_1.

DBN Initialization Finally, we need to initialize the DBN for $t = 0$, and provide the prior distribution $\mathcal{PD} = P(\bar{X}_0)$. This distribution is also extracted from the flow history: We search for traces of the respective flow model and create individual distributions for all the activities the flow has been started with at least once. For $\mathcal{F}_1$, this includes a_1, a_5 and a_6, and the distribution for $E_\alpha \in \epsilon_{a_1}$ could have the following values: $P(\mathit{wash}) = 0.01$, $P(\mathit{dry}) = 0.01$, $P(\mathit{write}) = 0.85$, $P(\mathit{fetch}) = 0.05$, $P(\mathit{disinfect}) = 0.01$ and $P(\perp) = 0.07$. In most of the cases the correct *writing* activity has been recorded. In some cases, *fetch* has been misinterpreted, while sometimes there was no meaningful evidence at all ($\perp$). The rates for the uncommon activities (*wash, dry, disinfect*) are even lower.

5.3 Clustered Particle Filtering

In order to exploit the knowledge encoded in the DBN for a specific flow model, a process called *inference* has to be executed. That is, the posteriori distribution of the variables (nodes) has to be calculated given real *evidence*. In our case, the evidence are the real context events received from the CMS in time slice t, and the inference is done by computing all the conditional probabilities for the variables in time slice $t + 1$. Exact inference is infeasible for complex DBNs like the ones generated from flows. Even more so, as this process is running in parallel to the flow execution: Whenever new evidence is available, the inference has to be done to get the probability distributions for the upcoming context events. Therefore, FlexCon uses a heuristic approach that is based on particle filters [19]. That is, we use a large number of random samples (the particles) from the distribution of the DBN at a certain time slice t and propagate them through the DBN to approximate the individual distributions associated with each node in the following time slice of the DBN. A particle filter approximates the exact distribution by generating a set of particles $N(\bar{X})$ for all random variables. The higher the number of particles the better the approximation of the real distribution. But the computation time grows linearly with the number of particles.

To propagate and calculate probabilities in the DBN the filter executes the following four steps. To initialize the filter, it first generates an initial particle set $N(\bar{X}_0)$ sampled from the prior distribution $\mathcal{PD} = P(\bar{X}_0)$ given by the DBN. In a second step each particle is propagated to the next time slice ($t = 1$ in this case) according to the distribution given by the conditional probability table. In the third step, the particles are weighted with the evidence available at the current time slice. Each particle is multiplied with the probability of the current observation. In the final step, the set of particles is resampled according to the weight of the individual particles. A detailed description of the basic principles has been published by Russel and Norvig [19].

We modified this standard algorithm as explained in the following, to accommodate it to the needs of FlexCon. The result is a *clustered particle filter* that is similar to the F3 filter presented by Ng et al. [21]. First of all, a single particle in FlexCon does not represent a full sample of $\bar{X}$ but only a sample of a subset of the variables $\bigcup_{E \in \epsilon_a} \chi(a, E)$, i.e. all variables of a single activity. Therefore, we call it clustered particle filtering, where each cluster can also be identified by $N(\bar{\chi}(a, \epsilon_a))$. This is an useful abstraction for a number of reasons. Each time slice in the DBN covers the completion of a single activity in the flow. Therefore, it is enough to process particles of that activity. All other particles are only propagated as they may be needed later on. This allows us to increase the total number of particles as the average processing load per particle is decreased. The unprocessed particles can be directly transferred to the same node in the next time slice, without the need for a dependency between these nodes.

For example, consider the trace $\mathcal{T}_1 = (a_1, a_6, a_3, a_4, a_6)$. After executing a_1, the particles from $\bar{\chi}(a_1, \epsilon_{a_1})_0$ are propagated to $\bar{\chi}(a_3, \epsilon_{a_3})_1$ since there is a transition (a_1, a_3), while $\bar{\chi}(a_6, \epsilon_{a_6})_0$ are just passed to $\bar{\chi}(a_6, \epsilon_{a_6})_1$, without further processing.

The second modification changes the propagation and weighting steps. Usually the full set of evidence, i.e. $P(\bar{\chi}(a', \epsilon_{a'})_{t+1} | \bar{X}_t)$, is available for propagating the particles in time slice t. As we only process the particles for a single activity a and only observe the received events for this activity as evidence, we can only rely on the conditional probability $P(\bar{\chi}(a', \epsilon_{a'})_{t+1} | \bar{\chi}(a, \epsilon_a)_t)$, instead. This means that we cannot use the evidence of events that have been observed "outside" of the current cluster $N(\bar{\chi}(a, \epsilon_a)_t)$. As a consequence we introduce an small error in the inference. However, the majority of $X \in \bar{X}$ will be independent from the variables in $\bar{\chi}(a', \epsilon_{a'})$, because there is no dependency defined by the flow. Therefore the introduced error is rather low and we actually discuss in Section 6 that not using this evidence makes FlexCon a bit more robust. Alternatively, it would also be possible to *sample* the evidence from the current distribution $N(\bar{X} \setminus \bar{\chi}(a, \epsilon_a))$ of the other activities, but this also introduces inference errors.

After the propagation phase, the actual observations (i.e. the received event instances) become available to the DBN. We can then weight the particles multiplying the number of particles $|N(\chi(a, E) = u)|$ for a specific event type u with the actual probability of the event type given by $I^e_E(u)$. Based on the computed weights all the particles for $\bar{\chi}(a, \epsilon_a)$ are resampled according to the distribution of the weighted particles.

The third modification is the actual processing of the received event instance e in order to decrease its uncertainty. This step is accomplished after the propagation of the particles and before the weighting. We compute the conditional probability weights for I'^e_E from the particles in $\chi(a, E)$, where the weight

$$p' = \frac{|N(\chi(a, E) = u)|}{|N(\chi(a, E))|}$$

for $I'^e_E(u)$ is just the relative particle frequency, as the distribution in the sample $N(\bar{\chi}(a, \epsilon_a))$ represents a sufficient approximation of the correct conditional probability distribution.

All probabilities $p = I_E^e(u)$ are added to the respective p' and the resulting distribution is normalized again, yielding $I_E''^e(u)$

Algorithm 1 Clustered Particle Filter Algorithm

```
    Input: $\mathcal{DBN} = (\bar{X}, \mathcal{TM}, \mathcal{PD}), a, e[]$
    if $N(\bar{X}) = \emptyset$ then
        $N(\bar{\chi}(a, \epsilon_a)) \leftarrow createInitialParticleSet(\mathcal{PD})$
    end if
 5: for all $e \in e[]$ do
        $weightEvent(e, I_E^e, \chi(a, E))$
        $weightParticles(N(\chi(a, E)), I_E^e)$
        $N(\chi(a, E)) \leftarrow resampleParticles(N(\chi(a, E)))$
    end for
10: $propagateParticles(N(\bar{\chi}(a, \epsilon_a)), \mathcal{TM})$
```

Algorithm 1 depicts the standard particle filter algorithm including the changes introduced by FlexCon. The input to the algorithm includes the $\mathcal{DBN}$, the currently completed activity a and the set of event instances $e[]$, a has received.

6 Evaluation

For our evaluation, we have generated flows according to a probabilistic pattern-based model [22] that has the same properties as the flows observed in the real-world hospital scenario. We do this to get a number of flows that is large enough to achieve statistical relevance. The flows we generate have the same average number of activities and the same structural properties. Essentially, the ratio between activities that have normal transitions and activities that are connected to other activities by constraints is equal.

Use of *flow patterns* [23] allows us to generate imperative flows based on structures commonly found in human-centric flows. We generate these flows and randomly add a respective portion of unconnected *constraint-based activities* (CBAs) to the flow. Next, we randomly generate constraints and use these to connect the CBAs to the imperative parts of a flow. Finally, the resulting flows are validated by generating traces from them. Flows that produce deadlocks (two or more activities blocking each other due to conflicting constraints) are discarded.

Overall, we generated 165 structurally different flows and 200 traces per flow for our evaluations.

The simulation has three important independent parameters. The first one is the *navigation threshold* t_n of the flow engine as defined in Section 4. For a higher navigation threshold the flow engine accepts less uncertainty in the context events it receives. We tested t_n from 0.4 to 0.6 in steps of 0.05.

The second parameter is the *average recognition rate arr* of the CMS. When a context event e is created in the CMS, *arr* is the average probability assigned to the correct event type in the distribution I_E^e by the CMS. The remaining probability $1 - arr$ is geometrically distributed to the other event types of the respective event type set E.

The *variance v* is the third simulation parameter. It represents the noise added to the distribution I_E^e created by the CMS. The probability of each event type $u \in E$ is varied by $\pm v/2$, and I_E^e is normalized again. We evaluated the system for variance values between 0.05 and 0.6 in steps of 0.05.

To assess the performance of FlexCon we use the *relative event improvement* and the *number of completed flows* as our two metrics. The relative event improvement r is defined as $r = I_E''^e(u)/I_E^e(u)$ for the correct event type u. If $r > 1.0$, then FlexCon was able to provide additional evidence for the occurrence of the correct event type u, and the flow engine has a higher chance of making the correct navigation decision.

The number of completed flows is simply the percentage of all traces that did complete their execution successfully. We did include the learning of the model in the simulations and the execution starts without a flow history. To put our system further into perspective, we directly compare the results with our previous measurement of the same metrics in FlowCon. Note that the flows in FlowCon are purely imperative. That is, activities are connected by transitions and there are no constraints that leave the decision about the ordering of the activities to the user. Thus, the task of FlowCon is much easier than that of FlexCon due to the additional flexibility of the flows.

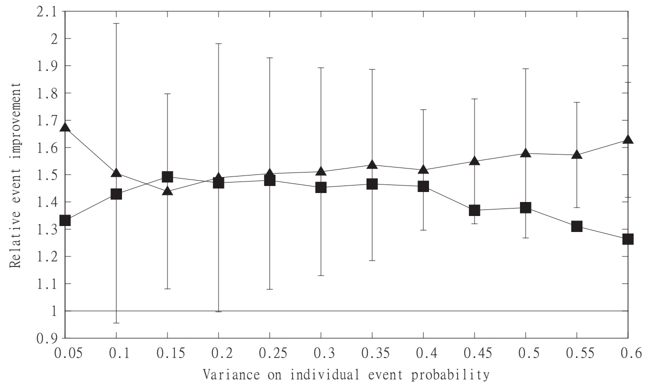
6.1 Results and Discussion

The evaluation results are depicted in Figure 3. We only show the results for $t_n = 0.4$ and $arr = 0.45$ for clarity. Furthermore, these conditions closely resemble the situation in the hospital and they can be compared best to our previous work.

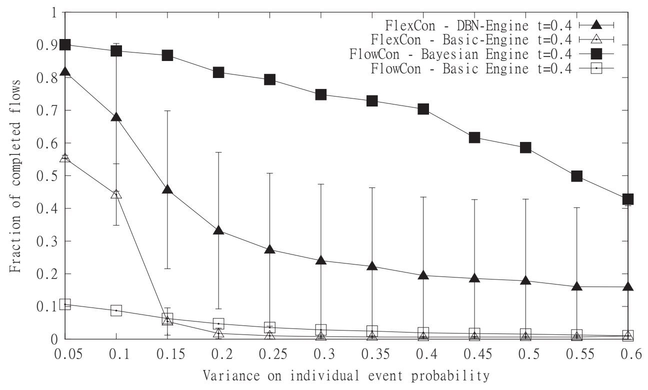
Figure 3(a) depicts the comparison of the relative event improvement rates for FlowCon and FlexCon. The average event improvement is better for almost all variance values. Even for the higher variances of $v \geq 0.4$, where the improvement of FlowCon declines, FlexCon is able to maintain a good improvement, mainly due to the changed method of accuracy improvement: While FlowCon uses all the observed event instances as evidence for calculating the probability of the current event, FlexCon only applies the evidence for the current particle for particle propagation, i.e. independently from other particles. When we misinterpret an event instance from a preceding node this has less impact on the particle filter, as only the propagated particles from this node are influenced, but not the particles from other preceding nodes. Where in FlowCon the whole conditional probability for the current event can be distorted, in FlexCon only a partial result suffers from the misinterpretation. However, if only one parent exists for a given node in the DBN, FlexCon is also sensitive to this kind of misinterpretation, leading to $r < 1.0$ making the result worse.

The high standard deviation for the event improvement on the flows can be explained by the flows' flexible structure. If two subsequently executed activities are not connected by a constraint or transition, we cannot improve the event in any way as there will be no connection in the DBN between the respective nodes. So according to the flow structure, we have a very high improvement for the dependent events but none for the independent ones.

Figure 3(b) shows the comparison of the flow completion rates, between FlowCon, FlexCon and the respective basic flow engines which do not take any action to decrease the event uncertainty. *FlowCon - Basic* and *FlexCon - Basic* simply execute the same



(a) Comparision of event improvement



(b) Comparision of flow completion

Fig. 3. Simulation Results - Comparision between FlowCon and FlexCon

flows without uncertainty reduction. Both basic systems fail at very low variance values. For $v \geq 0.15$ less than 6% of the flows can be completed successfully for both basic flow engines. The high values for the basic FlexCon flow engine compared to the basic FlowCon flow engine for $v = 0.05$ and $v = 0.1$ result from a changed method of generating the event instance distribution.

The FlexCon DBN-Engine manages to complete 45% of the flows at $v = 0.15$ and this performance decreases slowly for higher $v \geq 0.2$. It is still able to complete 20% of the flows at $v = 0.6$.

Again, the standard deviation on the number of completed flows is rather high, for the same reason as above. Some of the flows allow a very good event improvement leading to a reliable execution, after the training phase of the DBN is complete. Those flows (about 5% of the tested flows) exhibit an completion rate of well over 80% and are the main reason for the high standard deviation. Most of the flows are close to the average, and can complete their execution in about 30% of the cases.

7 Conclusions and Future Work

We have proposed FlexCon – a system that leverages the application knowledge encoded in workflows to make them more robust against inaccurate and noisy input data. FlexCon uses Dynamic Bayesian Networks and particle filters to reduce the uncertainty of the real-world context events received by *pervasive flows*. Our evaluations show that the uncertainty of an event received by a flow is reduced by 54% on average and the percentage of successfully completed flows is increased by 23-40%.

FlexCon is an important step towards applying flow technology as a part of pervasive systems. In real-world scenarios, found e.g. in the health care domain, users need to be supported in their activities without obstructing them. Thus, the flows need to automatically synchronize with their activities based on collected data such that users are not required to communicate with the flow explicitly. Especially in the health care domain, any explicit interaction (using touch screens etc.) may have severe implications in terms of hygiene.

The sensor data that is used to infer the current activity of a user is characterized by a high level of noise and inaccuracy. FlexCon offers a way to infer more reliable information from this data and, thus, render the respective flows more robust.

In our future work, we will investigate, if more sophisticated approaches to map the flow to a DBN yield a better event improvement. Furthermore we will optimize the number of particles used during the execution to speed up performance of the approach. Depending on the success we will adapt the prototype for a smart-phone, deploy in the hospital and study the usefulness. Furthermore we study the impact of a different uncertainty model on the recognition accuracy and the algorithm performance.

References

1. Leymann, F., Roller, D.: Production workflow: concepts and techniques. Prentice Hall PTR (2000)
2. Dadam, P., Reichert, M., Kuhn, K.: Clinical Workflows - The Killer Application for Process-oriented Information Systems? In: Proc. 4th Int'l Conference on Business Information Systems, Springer (April 2000) 36–59
3. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development **23**(2) (2009) 99–113
4. 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)
5. 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
6. Kunze, K., Lukowicz, P.: Dealing with sensor displacement in motion-based onbody activity recognition systems. In: Proceedings of the 10th international conference on Ubiquitous computing. UbiComp '08, New York, NY, USA, ACM (2008) 20–29
7. Bahle, G., Kunze, K., Lukowicz, P.: On the use of magnetic field disturbances as features for activity recognition with on body sensors. In: Proceedings of the 5th European conference on Smart sensing and context. EuroSSC'10, Berlin, Heidelberg, Springer-Verlag (2010) 71–81

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. 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
10. 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
11. 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
12. Wieland, M., Käppeler, U.P., Levi, P., Leymann, F., Nicklas, D.: Towards Integration of Uncertain Sensor Data into Context-aware Workflows. In: Tagungsband INFORMATIK 2009 Im Focus das Leben, 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Lübeck, Lecture Notes in Informatics (LNI) (September 2009)
13. Adam, O., Thomas, O.: A fuzzy based approach to the improvement of business processes. In: First International Workshop on Business Process Intelligence (BPI05). (September 2005) 25–35
14. Urbanski, S., Huber, E., Wieland, M., Leymann, F., Nicklas, D.: Perflows for the computers of the 21st century. In: Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on. (March 2009) 1 –6
15. Wolf, H., Herrmann, K., Rothermel, K.: Modeling dynamic context awareness for situated workflows. In R. Meersman, P.H., (Eds.), T.D., eds.: OTM 2009 Workshops. Volume 5872 of LNCS., Vilamoura, Springer-Verlag Berlin Heidelberg (November 2009) 98–107
16. Pesic, M., Schonenberg, H., van der Aalst, W.M.: Declare: Full support for loosely-structured processes. Enterprise Distributed Object Computing Conference, IEEE International **0** (2007) 287
17. Aalst, W.M., Adams, M., Hofstede, A.H., Pesic, M., Schonenberg, H. In: Flexibility as a Service. Springer-Verlag, Berlin, Heidelberg (2009) 319–333
18. Giannakopoulou, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: In Proceedings, International Conference on Automated Software Engineering (ASE01), IEEE Computer Society (2001) 412–416
19. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd edition edn. Prentice Hall (2002)
20. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, UNIVERSITY OF CALIFORNIA, BERKELEY (2002)
21. Ng, B., Peshkin, L., Pfeffer, A.: Factored particles for scalable monitoring. In: In Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann (2002) 370–377
22. 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.
23. 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