Institute of Architecture of Application Systems

An Event-model for Constraint-based Person-centric Flows

Tobias Unger, Hanna Eberle, Frank Leymann, Sebastian Wagner

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings {INPROC-2010-101,
    author    = {Tobias Unger and Hanna Eberle and Frank Leymann and Sebastian
                 Wagner},
    title     = {An Event-model for Constraint-based Person-centric Flows},
    booktitle = {Proceedings of the 2010 International Conference on Progress
                 in Informatics and Computing (PIC-2010)},
    year      = {2010},
    pages     = {927--932},
    doi       = {10.1109/PIC.2010.5687886},
    publisher = {IEEE Computer Society}
}
```


Universität Stuttgart
Germany

An Event-model for Constraint-based Person-centric Flows

Tobias Unger, Hanna Eberle, Frank Leymann, and Sebastian Wagner
University of Stuttgart
Universitaetsstr. 38, 70569 Stuttgart, Germany
Email: {firstname.lastname}@iaas.uni-stuttgart.de

***Abstract*—Over the past years research in pervasive computing has demonstrated the potential of context-aware and proactive technologies for improving human work performance and to ensure that people act compliant according to predefined regulations. Human work can be structured into tasks, whereas a task is representing an atomic human work entity. A person-centric flow is an IT-representation of the flow of activities an individual person is performing. For example the daily care schedule of a nurse can be understood as the person-centric flow of the nurse. To be able to effectively guide a person in a complex and highly dynamic work environment and to react on possible deviations from the flow the supporting system is required to be aware of the state of the person-centric-flow. Beside guidance the flow information can be utilized to check compliance of a persons flow with prescribed sequences of operation. In this paper we propose a constraint-based workflow model for person-centric flows and an event-model which can be used to inform applications about the state of these flows.**

***Keywords*—Human-centric BPM, Constraint-based Workflow**

I. Introduction

A Person-centric Flow is an IT-representation of the flow of activities an individual person is performing [1]. For example the daily care schedule a of nurse can be understood as the person-centric flow of the nurse. Mostly, these activities are planned and created by the hospital's healthcare documentation and planning system. The IT-representation of such an activity is called human task. People are informed about their tasks using worklists.

The person-centric flow is created by enriching the worklist with flow information, which can be recommendations or obligations. The person-centric flow can be synchronized with what the person is doing either explicitly by ticking the tasks on the worklist or by observing the person using activity sensing [2]. The other way round, person's can be provided with recommendations and instructions how to execute their tasks based on the flow information which help people e.g. to save resources or time. This can be done e.g. by calling their attention to possible errors or to the fact, that the chosen ordering of the tasks is prohibited. Ambient guidance strategies provide means to direct and correct humans if necessary, e.g. if a nurse forgets an important procedure [3]. To be able to implement ambient guidance the ambient guidance system must be aware of the work a person has to perform and how this work should be done. Therefore, the person-centric flow opens great opportunities to support people in doing their work and to ensure compliance to certain regulations. The person-centric flow can be utilized to check whether a person deviates from her person-centric flow and whether this deviation is tolerable or not. Applications can be informed about the violation of the person's behavior from her person-centric flow using events. Events are also generated in case a person deviates from her flow. Imagine a nurse has a person-centric flow which prescribes as first step to measure the blood pressure of patient A, then to disinfect her hands, and, finally, to measure the blood pressure of patient B. If the nurse decides to take the blood pressure of person B directly after measuring blood pressure A without disinfecting her hands between these to taks, the nurse deviates from her person-centric flow in an intolerable way and the ambient guidance system is informed by a *violation event*. The violation information can be used to inform the nurse that she has to disinfect her hands. Additionally, also tolerable deviations can be detected. A tolerable deviation would be to measure blood pressure of patient B, then to disinfect her hands, and, finally, to measure the blood pressure of person A. Also in this case, the ambient guidance system can be informed about the deviation by a *deviation event*. Such an information can be used e.g. for providing the nurse with the health record of person B instead of providing her with the health record of person A.

The contribution of this work is divided in two parts: First we present a constraint-based workflow model for person-centric flows. However, the algorithms for enriching worklists with flow information in order to create the person-centric flow are out of this scope of this work. Appropriate algorithms are presented in [1], [4]–[6]. The second contribution is an event-model for person-centric flows, which can be used to inform applications about the state of the person-centric flow. Applications, e.g. the ambient guidance system, can register for these events. Afterwards, the flow system informs the application about the state of the person-centric flow. For example the flow system sends an event to the ambient guidance system in case the nurse deviates from the prescribed flow.

The reminder of the paper is as follows: Section II introduces the concept of person-centric flows. In Section III the architecture for distributing the events of the person-centric-flow is presented and, finally, in Section IV the events and the determination of these events is presented. We conclude our work with a discussion of our approach based on related work approaches in Section V and with a summary of our results in Section VI.

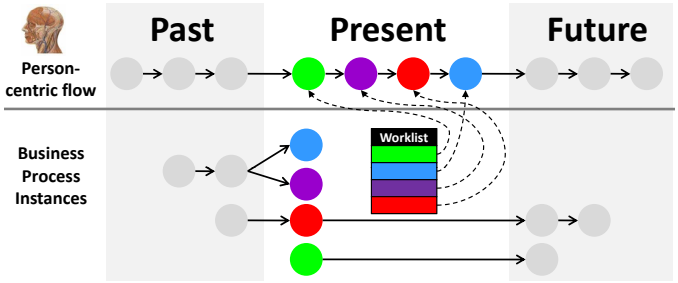

Figure 1. Person-centric Flow

II. Person-centric Flows

Definition 1 (Person-centric flow): A person-centric flow defines a partial ordering over a set of tasks which have to be performed by one single person. Tasks of a person-centric flow can be classified in three tenses: past tasks, present tasks, and future tasks. Past tasks are either completed correctly or incorrectly and their ordering is known. Present tasks are tasks that are currently presented in a person's worklist. Their ordering is planned by the person but can change dynamically. Future tasks are tasks which are assumed to be executed in the future. Both the set of future task as well as their ordering may change dynamically.

Fig. 1 shows a simple graphical representation of a person-centric flow. Please note that present and future tasks can be mixed up within a person-centric flow. Only the task that is actually executed must be included in the set of present tasks. Since a person-centric flow is formed implicitly in a person's mind it must be predicted by the WfMS in order to utilize the ordering information. A person cannot be demanded to tell the WfMS its actual flow. Since tasks appear and disappear in a high frequency, this would be an additional stress factor. As a consequence predictions may be wrong. Furthermore, the person-centric flow paradigm is partly contrary to the existing workflow paradigm. For example a person-centric flow has no prescribed flow model as the set of tasks changes dynamically and so does the ordering of the tasks. Many control flow patterns like loops are not needed in person-centric flows since each task is executed once. A single instance of a person-centric flow is associated to one person. In this paper we rely on declarative workflows in order to describe a person-centric flow of a person [7].

A. A Formal Definition for Person-centric Flows

In this Section we present our formalization of person-centric flows. The formalization bases on the formalization presented in [8]. Let $\mathscr{P}$ be the set of all persons. $\mathscr{TM}$ denotes the universe of all available task models. The runtime instances of task models are denoted as the set $\mathscr{TI} = \{\mathscr{TM} \times \mathbb{N}\}$. Each task instance has a state, where the set of possible states is denoted as $taskState : \mathscr{TI} \rightarrow \{\perp, activated, running, completed\}$. Task instances with a task state $\perp$ denote virtual task instances, which are going to be created in the future, but nevertheless they are an important part of the person-centric flow determination. Each task instance has exactly one person assigned to it, which is defined by the map: $staffAssignment : \mathscr{TI} \rightarrow \mathscr{P}$. If a task instance execution is started and the task state changes from activated to running the starting time for that task instance is set. The starting time of a task instance is later retrieved by the map: $startingTime : \mathscr{TI} \rightarrow (\mathbb{N} \cup \{\perp\})$. If the task instance changes to the state completed the completion time is set, which can also be retrieved by a map: $completionTime : \mathscr{TI} \rightarrow (\mathbb{N} \cup \{\perp\})$. The set of all constraints is denoted by the set $\mathscr{C}$. Constraints are defined on task instances, which are assigned to a constraint by the map: $tasksOfConstraint : \mathscr{C} \rightarrow 2^{\mathscr{TI}}$. There exist several types of constraints, which can be classified in three different classes. Unary constraint types define restrictions on one single task instance, e.g. how many times this instance has to be executed. If the *init* constraint was defined on a task model, an instance of this task model must be the first task that is executed during workflow execution. The *last* constraint on the other hand defines that an instance of the task model where the constraint was defined on must be the last task that is executed during workflow execution. Constraints types of the choice constraint class are used to specify, that a subset of a set of task instances has to be chosen for execution. Relation constraints types define the execution order of the instances of two task instances. The *A precedence B* constraint is an example for this constraint class. A more detailed description of the constraint types can be found in [7], [9].

A person-centric flow is denoted as follows: A person-centric flow of a person $p \in \mathscr{P}$ at an observable point in time $i \in \mathbb{N}$ is a tuple $PCF_{p,i} = (PTI_{p,i}, HTI_{p,i}, C_i, c_{strength}, c_{type}, m)$, where $PTI_p = \{ti | staffAssignment(ti) = p \wedge taskState(ti) \in \{activated, running\}\}$ denotes the present task instances of a person and $HTI_p = \{ti | staffAssignment(ti) = p \wedge taskState(ti) = completed\}$ the already completed task instances or history task instances. $C_i \subseteq \mathscr{C}$ denotes the set of the currently existing and to be satisfied constraints. The C_i evolves over time, always depending on the present task instances and the history task instances, since the tasks involved in C_i must be of the present task instances or history task instances of the $PCF_{p,i}$. For each constraint in C_i a strength and a type is specified. The strength notes, whether a constraint is optional or mandatory. Optional constraints are usually used for guidance and mandatory constraints are needed to ensure certain qualities and requirements: $c_{strength} : C_i \rightarrow \{optional, mandatory\}$. The type of a constraint obtains one of the three options intention, guidance or enforcement. Intentional constraints are constraints that are used to be able to follow the flow a person has in mind. Guidance constraints are constraints that can be used to include special guidance constraints (e.g. a recommendation to measure the heart rate before measuring the blood pressure) within the person-centric flow. Enforcement constraints are used e.g. to support a proactive ambient guidance for constraints which have to be satisfied (e.g. that a nurse hast to disinfect her hand after washing a patient). Therefore the map $c_{type} : C_i \rightarrow \{intention, guidance, enforcement\}$ is defined in order to assign a constraint with a strength. The constraint set C_i of a $PCF_{p,i}$ must hold, that if a constraint c in C_i is a constraint of the intensional type, the strength of the same constraint must be

optional, since intensional constraints are supporting constraints but not constraints that are enforced. On the other hand, if a constraint c in C_i is an enforcement constraint the strength of the same constraint must be mandatory. We define $C_i = \{c \in \mathscr{C} | (tasksOfConstraint(c) \in PTI_{p,i} \cup HTI_{p,i})$ and $(c_{strength}(c) = optional$ if $c_{type}(c) = intention)$ and $(c_{strength}(c) = mandatory$ if $c_{type}(c) = enforcement)\}$ The mode of the person-centric flow is set by $m \in \mathscr{M}$.

B. Person-centric Flow Execution

A person-centric flow gets executed by the nurse performing her tasks. If the nurse starts a task the task in the person-centric flow gets triggered and set to state running. The work a nurse is performing is evaluated against the constraint set of the intended person-centric flow. Depending on that evaluation, the person-centric flow must be either adapted to the real world, or if the nurse is violating mandatory constraints, the system generates corrective guidance events. The evaluation of the person-centric flow is defined as follows. Since the $PCF_{p,i}$ is executed and its definition changes over time, the constraint set is evaluated against a time depending constraint set with a time depending execution state. The execution state of a $PCF_{p,i}$ for a person $p \in \mathscr{P}$ and a point in time $i \in \mathscr{N}$ is defined by the task states. The history task instance set $HTI_{p,i}$ and the running task instances of $PTI_{p,i}$ are evaluated against the constraints. This joined task instances set is called satisfaction task instances set $STI_{p,i}$, with $STI_{p,i} = HTI_{p,i} \cup \{ti | ti \in PTI_{p,i} \wedge taskState(ti) = running\}$. All the tasks of the $STI_{p,i}$ define are running or completed and therefore the task starting time is specified. Therefore the $STI_{p,i}$ can be used for constraint evaluation, hence there already exists a ordering between the tasks this set defined by the task instances starting time. The ordering of the executed task instances denotes the execution trace of the $PCF_{p,i}$. The remaining task instances are the activated or open task instances of $PCF_{p,i}$, which are denoted as $OTI_{p,i} = \{ti | ti \in PTI_p \wedge taskState(ti) = activated\}$. Based on the $OTI_{p,i}$ we are able to determine, whether it will be possible to satisfy C_i, if C_i is violated by the $STI_{p,i}$. A violated C_i is can be satisfied, if there exists a partial execution trace or a partial ordering of $OTI_{p,i} \cup STI_{p,i}$, which satisfies C_i. Therefore we define the constraint evaluation procedure for a $PCF_{p,i}$ at a certain point in time i as follows:

$$eval(STI, OTI, C) = \begin{cases} satisfied & if\ STI_P \models C \\ temporarily violated & if\ (STI \not\models C) \wedge (\exists t \in 2^{OTI} : STI \cup t \models C) \\ violated & otherwise \end{cases}$$

C. Person-centric Flow Generation Algorithms

Since the person-centric flow may change dynamically according to the current situation, the constraints must be re-generated continuously. The prediction algorithms create a constraint set based on the present tasks PTI_p and past tasks HTI_p. Formally, a prediction algorithm is a function $prediction(PTI_h, PTI_p) \subseteq \mathscr{C}$ returning the actual valid constraint set for the set of tasks.

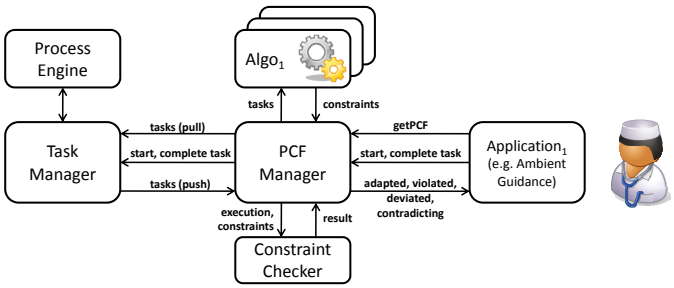

Figure 2. Architecture

Since we are not able to capture the person-centric flow a person has in mind, we need to find algorithms to determine the task orderings. Especially history based algorithms are promising since people often have behavior patterns which can be detected by history-based algorithms. Also algorithms operating e.g. on task deadlines without considering the history (e.g. scheduling algorithms) are appropriated [4]–[6].

III. Architecture

The major idea of our approach is to utilize the knowledge about the person-centric flow and its state in order to improve applications like ambient guidance. Fig. 2 shows our overall architecture. The central element is the person-centric flow manager, which receives the constraints from the plugged in constraint generation algorithms. We argue that according to the scenario different constraint generation algorithms will be necessary (cf. Sec. II-C). For example a healthcare scenario differs completely from a delivery scenario. Tasks are managed in a task manger which executes the business process in cooperation with the process engine.

A constraint checker component evaluates the execution against the current person-centric flow as well as the person-centric flow model against itself. The events that are produced by the constraint checker component are utilized by the registered ambient guidance components. For example an ambient guidance system might use the person-centric flow events to provide the person with directions and guiding the person pro-actively through all her work. If a person deviates from the person-centric flow or if she even violates the person-centric flow an event is thrown by the constraint checker component. The information contained in this event can be utilized by the ambient guidance to adapt the guidance. For example if a person starts another task than predicted, the ambient guidance attune to the new situation by updating the guidance information presented to the user. Additionally, after an execution violation a new generation of the person-centric flow is triggered. In the case that the person-centric flow is adapted or the model is self contradicting the constraint checker throws either an adaptation event or a contradiction event. The adaptation event indicates that the plans of the person have changed, which facilitates the reconfiguration of the ambient guidance component in order to provide the person with information about tasks according to the changed plans.

IV. Event Model for Person-centric Flows

In order to guide people effectively we need information about the intended person-centric flow of a person. This encompasses the list of task instances that have to be performed by a person and the predicted constraints that have to be met by these task instances. Moreover, information about the execution state of the person-centric flow have to be gathered and published (refer to *Execution-related Events* below), e.g. whether the execution is deviating from the predicted person-centric flow or even violating the constraints. In order to avoid that applications have to periodically request for state changes in the person-centric flow (e.g. whether the flow was violated), the person-centric flow manager (PCF Manager) sends events to the applications if the state of the flow has changed. In addition, there are also events sent to the PCF Manager. These are real world events (e.g. if a nurse starts blood pressure measurement) that are captured using activity recognition. After the PCF Manager has received such events it triggers the Constraint Checker that checks that the person behaves as expected by verifying that the actual valid constraints are met. If she does not behave as expected an event is sent to the application by the PCF Manager. The advantages of using events are that we can evaluate the constraints in a centralized way and that we are able to inform other components (e.g. the application) that are interested in these events almost in real time. Furthermore, it prevents the person-centric flow manager from being overburdened by answering continuous polls from the applications.

Deviations and violations are detected by the Constraint Checker component. It verifies that no inconsistencies emerge between the constraints and it also ensures that the constraints are met during the execution of the person-centric flow. The Constraint Checker is always triggered when tasks change their state to *running* or *completed* and when the constraint or task model set has changed. Depending on the outcome of the verifications the Constraint Checker can raise the following events:

a) Model-related Events:

- *Adapted*: A person-centric flow is adapted, if the set of task and/or the set of constraint changes.
- *Contradicted*: It may happen that new constraints contradict with already existing constraints. If this is the case the flow is in "contradicted" state. For example if the intention of a nurse is to wash a patient before taking the heart rate this may contradict to a rule saying that the tasks have to be done the other way round. In this case for example an intention constraint would contradict to an enforcement constraint.

b) Execution-related Events:

- *Satisfied*: A person-centric flow is satisfied, if all constraints evaluate to true. In other words the person behaves as predicted and her behavior causes no violation of e.g. a security rule.
- *Deviated*: A person-centric flow is in state deviated, if the person deviated from the predicted behavior. For example tasks are executed in a different order than predicted.
- *Violated*: A person-centric flow is in state violated, if the person violates for example a security constraint. Technically spoken this means that at least one of the mandatory constraints is violated.

It has to be clearly stated that our approach only informs the applications about deviations or violations of the person from the person-centric flow. We do not prevent a person from starting tasks which cause a violation of the person-centric flow. This is due to several reasons: Actions are executed in the real world. Mostly, if a violation of the person-centric flow is detected, the person has already started working on a task. Generally, two reactions are possible. On the one hand the person is informed about the violation and can decide whether to stop or continue the work. On the other hand, due to changing situations, the constraints are changing so that working on the task would no longer cause a violation of the person-centric flow. In the latter case, the violation disappears automatically as soon as the constraints are evaluated the next time.

A. Model-related Events Generation Implementation

In order to generate model-related events the Constraint Checker has to validate the consistency of the constraints models to ensure that there exist no contradictions [7] between them. In the following an overview is given that describes how the Constraint Checker detects contradictions. A more detailed description of these steps can be found in [7]. If any inconsistency between two or more constraints was detected the Constraint Checker sends a "Contradicted" event. Since tasks and constraints are generated dynamically, for each task a task model is created at runtime. These task models are required by the constraint validation mechanisms that are described here. To validate relation, *init*, *last* and negation constraints [7] a constraint network is created and validated. In the following it is described how this is done.

The relation constraints are transformed to interval relations of the interval algebra [10] where each task model is represented by an interval. For instance if a *response* constraint is defined between task models `A` and `B` it is transformed to the interval relation $A\{before, meets\}B$. In the terms of interval algebra this means, that interval (task) *A* has to appear either immediately (indicated by the *meets* relation) or anytime before *B* (indicated by the *before* relation). The interval relations that are created from the relation constraints form a constraint network like the one that is illustrated in Fig. 3.

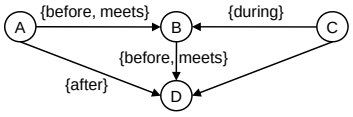

Figure 3. Example Constraint Network

In [10] and [11] reasoning algorithms were introduced that can be applied on constraint networks. These reasoning algorithms infer transitively the relations between all intervals and

discover contradictions between them. For instance in Fig. 3, it can be inferred from the relations "C has to be executed during the execution of B" and "B has to be executed before D" that C must be executed before D. Moreover, it can be also inferred that a contradiction exists in the network since A can not be performed after D but before B. If such contradictions are detected we can conclude that there exist inconsistencies between the relation constraints (including the *init* and *last* constraint) were the constraint network was created from. In [9] a comprehensive overview is provided how to detect inconsistencies between constraints.

B. Execution-related Events Generation Implementation

To emit execution-related events the Constraint Checker verifies during the execution of $PCF_{p,i}$ if the constraints are met. If all constraints are met $PCF_{p,i}$ is in the state *satisfied* and a "Satisfied" event is generated. On the other hand, if one or more of the constraints are violated $PCF_{p,i}$ is in the *temporarily violated* or in the *violated* state. If the violation of an optional constraint was determined (temporarily or permanent) the "Deviated" event is emitted. If a mandatory constraint was violated the "Violated" event is generated. To perform the verification the Constraint Checker uses the procedure $eval(STI, OTI, C)$. In the following it is described for each constraint how the procedure determines if the constraint was met.

***Init* constraint:** An init constraint is

- *satisfied*, if an instance of the task model, where the constraint was defined on, is the first task of the $PCF_{p,i}$ that was executed;
- *violated*, if an instance of another task model, i.e. a task model where the constraint is not defined on is the first task of the $PCF_{p,i}$ that was started;
- *temporarily violated*, if $PCF_{p,i}$ was started but no task was executed yet, i.e. an instance of the task model where the *init* constraint was defined on can still be the first task that is executed.

***Last* constraint:** A last constraint evaluates to

- *satisfied*: If an instance of the task model that is associated with this constraint is executed and no other task of the $PCF_{p,i}$ is in the state *running* anymore.
- *violated*: This constraint can not cause the $PCF_{p,i}$ to become violated. Even if the user executes an instance of a task model that is not associated with this constraint she has always the possibility to start the task that is associated to the *last* constraint when no other task is in the state *running*.
- *temporarily violated*: If an instance of the task model was not started as last task of the $PCF_{p,i}$.

Existence constraints: It has to be simply counted in the $STI_{p,i}$ how many instances of the task models, where this constraint was defined on, are in the *running* or *completed* state.

- *satisfied*: If the number meets the lower and upper bound that was defined by the constraint.
- *violated*: If the upper bound concerning the allowed number of *running* or *completed* instances of the task model was exceeded in $STI_{p,i}$.
- *temporarily violated*: If the lower bound concerning the minimum required instances of *running* or *completed* instances of the task model was not met in $STI_{p,i}$ yet.

Choice constraints: The *choice* constraint defines that from a set of task models a certain number K of them has to be instantiated and executed. It can be simply checked in $STI_{p,i}$ if instances of the K different task models appear there.

- *satisfied*: If K or more instances of different task models from the set are a members of $STI_{p,i}$.
- *violated*: The *choice* constraint can not violate $PCF_{p,i}$.
- *temporarily violated*: If less than K instances of different task models from the set are members of $STI_{p,i}$.

Relation constraints: To verify that the relation constraints were met we are utilizing again the constraint network that was introduced above. The Constraint Checker determines for each task that is put into the *running* and *completed* state if the relations of this task to the other tasks in $STI_{p,i}$ corresponds to the interval relations between the task model where the task was created from and all other task models in the constraint network. In order to determine the partial order between the task that was started or completed and the other tasks we express the relation between the tasks with the point algebra [11] [12] (this algebra defines a partial order between the start and end points of a task pair). This enables us to determine the interval relations between the task and the other tasks. Then it is simply checked if the determined interval relations match with the interval relations in the constraint network. The constraint network in Fig. 3 defines for instance that a *during* interval relation must hold between the tasks C and B. If $STI_{p,i}$ would contain the information that task B was completed before D the *during* constraint that is represented by the interval relation would have been violated. However, this approach can only determine if permanent violations have occurred.

- *satisfied*: If the relations of the task that was started or completed and the other tasks in $STI_{p,i}$ match with the interval relations that were defined in the constraint network.
- *violated*: If there are any relations between the task that was started or completed and the other tasks in $STI_{p,i}$ that does not correspond to the interval relations defined by the constraint network.
- *temporarily violated*: If there is a sequential relation constraint defined between the task models A and B and an instance of A is in the *completed* state but an instance of B has not been executed yet. Or if a parallel relation constraint was defined between A and B and the instances of these models have not been completed yet. For instance if a *during* constraint was defined between A and B and an instance of A was started after an instance of B but both tasks were not completed yet.

Negation constraints: As mentioned before negation constraints are transformed to the interval relations that represent

the constraint that is negated. If a task is put into *running* or *completed* state it is checked if $STI_{p,i}$ contains the forbidden relation.

- *satisfied*: If there exists no relation in $STI_{p,i}$ between the task and the other tasks that is forbidden by the negation constraint.
- *violated*: If there exists any relation in $STI_{p,i}$ between the task and the other tasks that is forbidden by the negation constraint.
- *temporarily violated*: Negation constraints can not violate $PCF_{p,i}$ temporarily.

V. Related Work

In this work we proposed an event-model for person-centric flows to be able to apply the concept of person-centric flows for ambient guidance [3]. Our person-centric flow language [7] is based upon the interval relations of Allen's interval algebra [10] and belongs to the class of declarative workflow languages [8], [13], [14]. In contrast to other existing declarative languages the person-centric flow language supports constraints to add support for temporal restrictions, additionally. Evaluation algorithms to determine the executions' compliance with the restrictions of the allowed behavior as modeled in the person-centric flow can be analogously defined to the algorithms as presented in [15]. To allow the person-centric flow to be driven by the outside world, we introduced an event model. In case the execution of the outside deviates or even violates the prescribed activity ordering an event is pushed back to, e.g., the application. Event models in the workflow domain are not new, even though the application of the event-model for person-centric flows as presented in this work is different, e.g., there exists an event model for the workflow language BPEL as presented in [16], which specifies events that are created during the execution of a BPEL workflow. WS-HT [17] specifies a model and events for the human task execution. This human task model and event specification can be used to extend the event model presented in this work to provide a more fine-granular view on the task states and execution.

VI. Conclusion

In this paper we introduce the person-centric flows as a supplementing concept to existing workflow technology. Our person-centric flow model is based on constraints and, therefore, provides a large degree of flexibility. A person has the freedom of decision to decide the execution order of her tasks within the boundaries of the constraints defined by the person-centric flow. In addition to the person-centric flow definition we provide a revised execution architecture and event model. Thus, deviations or violations of a person executing her person-centric flow can be detected and distributed using events. Our architecture fulfills the requirement to reduce communication to a minimum. We claim that the knowledge provided by a person-centric flow can be utilized to establish for example ambient guidance. Our prototype bases on *bangkok* a BPEL4People compliant task manager [18]. In order to generate the events described above, we extend *bangkok* with a person-centric flow manager. In future work we will investigate in validating our prototype in a real world hospital scenario.

Acknowledgment

This research was supported by EU FP7 research grants 213339 (ALLOW) and 216917 (MASTER).

References

[1] T. Unger, H. Eberle, and F. Leymann, "Research Challenges on Person-centric Flow," in *ZEUS*, 2010. [Online]. Available: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-563/paper12.pdf

[2] K. S. Kunze, F. Wagner, E. Kartal, E. M. Kluge, and P. Lukowicz, "Does Context Matter? - A Quantitative Evaluation in a Real World Maintenance Scenario," in *Pervasive*, 2009, pp. 372–389.

[3] G. Kortuem, F. Kawsar, and B. A. Takrouri, "Flow Driven Ambient Guidance," in *PerCom 2010*, 2010.

[4] H. Schonenberg, B. Weber, B. F. van Dongen, and W. M. P. van der Aalst, "Supporting Flexible Processes through Recommendations Based on History," in *BPM*, 2008.

[5] R. Han, Y. Liu, L. Wen, and J. Wang, "A Two-Stage Probabilistic Approach to Manage Personal Worklist in Workflow Management Systems," in *OTM Conferences (1)*, 2009, pp. 24–41.

[6] J. Petzold, F. Bagci, W. Trumler, and T. Ungerer, "Comparison of different methods for next location prediction," in *Euro-Par*, 2006.

[7] F. Leymann, T. Unger, and S. Wagner, "On Designing a People-oriented Constraint-based Workflow Language," in *ZEUS*, 2010.

[8] M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. van der Aalst, "Constraint-based workflow models: Change made easy," in *OTM Conferences (1)*, 2007.

[9] S. Wagner, "A Concept of Human-oriented Workflows," Diploma Thesis, University of Stuttgart, Germany, January 2010. [Online]. Available: http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-2987&engl=1

[10] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[11] M. B. Vilain and H. A. Kautz, "Constraint propagation algorithms for temporal reasoning," in *AAAI*, 1986, pp. 377–382.

[12] P. van Beek, "Exact and approximate reasoning about qualitative temporal relations," Ph.D. dissertation, 1990.

[13] M. Pesic, "Constraint-based workflow management systems: Shifting control to users." Ph.D. dissertation, Eindhoven University of Technology, 2008.

[14] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Computer Science - R&D*, vol. 23, no. 2, pp. 99–113, 2009.

[15] R. Lu, S. W. Sadiq, V. Padmanabhan, and G. Governatori, "Using a temporal constraint network for business process execution," in *ADC*, 2006, pp. 157–166.

[16] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszek, and F. Leymann, "BPEL Event Model," University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, University of Stuttgart, Institute of Architecture of Application Systems, Technical Report Computer Science 2006/10, November 2006. [Online]. Available: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2006-10&engl=1

[17] OASIS, *Web Services Human Task Specification Version 1.1, Committee Draft 06*, 2009. [Online]. Available: http://www-128.ibm.com/developerworks/library/specification/ws-bpel4people/

[18] T. Unger and S. Wagner, *Project Bangkok*, 2010. [Online]. Available: http://code.google.com/p/projectbangkok/