Modeling Dynamic Context Awareness for Situated Workflows

Hannes Wolf, Klaus Herrmann, and Kurt Rothermel

Institute of Parallel and Distributed Systems,
Universitätsstraße 38,
D-70569 Stuttgart, Germany
{forename.surname}@ipvs.uni-stuttgart.de

Abstract. A major challenge for pervasive computing is to support continuous adaptation of applications to the behavior of the user. Recent research has adopted classical workflows as alternative programming paradigm for pervasive applications and approaches for context aware workflow models have been presented. However the current approaches suffer from the low flexibility of classical workflow models. We present a solution that allows attaching workflows to real-world objects and defining relevant context dynamically in relation to those objects. The benefits are a dynamic, yet simple modeling of context constraints and events in pervasive workflows and a greatly reduced amount of context information that must be provided to the workflow.

1 Introduction

The great challenge of pervasive computing is the unobtrusive support of users in diverse tasks. New application paradigms have been developed to tackle this challenge. A key source of information for these applications is context information. A special kind of context aware mobile applications are situated applications introduced by Hull et al. [1]. This kind of applications is able to detect, interact and respond to the local (physical) context of the application itself or the user. This way the all the context information in the local environment is available for the application and it is up to the application programmer which context information is accessed.

A context system that hosts context aware applications must be able to provide all the context information that any application is interested in, in a best effort manner. We refer to this as static context provisioning. We suggest that the Adaptable Pervasive Flow (APF or simply *flow*) [2] is a suitable programming paradigm for optimizing the provisioning of context information. Our research on flows is conducted in the ALLOW project [1] funded by the European Union.

[1] This research has been supported by 7th Framework EU-FET project 213339 – ALLOW.

An Adaptable Pervasive Flow is a far reaching extension of the classical workflow paradigm that tackles adaptation, user interaction, security issues and context recognition. This paper contributes to functional specification of APFs. The classical workflow paradigm is usually used for programming in the large and allows orchestration of a set of activities on a high level of abstraction. Activities in APF can call another flow or (web) service like in the classical workflow model or represent a task that is directly executed by a human user.

In this paper we focus on three of the new aspects of the APF: First we show in detail how we extend the classical workflow model in order to make flows situated. Secondly we provide a modeling concept that allows the flow modeler to dynamically define what kind of context information is relevant during the execution, exploiting the situatedness. We show that our modeling concept enables the underlying context provisioning system to dynamically provide only the relevant context information to a single flow application. We call this dynamic context provisioning. The third contribution is a constraint and event handling mechanism that enables a flow directly to react to changes of the relevant context information appropriately.

The rest of the paper is structured as follows. In section 2 we introduce our system model and the running example. The related work is discussed in section 3. We then present in detail our modeling extensions in conjunction with a scenario walkthrough in section 4. Finally, we discuss our approach in section 5 and conclude our work in section 6.

2 System Model and Application Scenario

Workflow models are modeled using a workflow modeling language. A common wide spread and standardized language is the Business Process Execution Language (BPEL). BPEL is based on XML, which easily allows extensions. In order to execute a workflow model, an instance of that model must be created and run on a workflow engine (WFE). Flows are considered mobile and can migrate from one WFE to another in order to optimize their execution. Classical workflows are usually created for scenarios that exhibit at least some degree of repetition. Following that we assume that some of the running flow instances in our system are created from the same flow model. Because situated flows must be context aware, our system provides a context access layer. The nodes that host this layer are connected in an overlay network with undirected links that significantly vary in bandwidth and latency. The representation of stored context information is based on our object-oriented *entity model*, which will be introduced in section 4.1 in more detail. We assume a system that hosts a large number of flows and spans a wider geographical region.

In order to show the feasibility of our system, we will apply our concepts to an example real-world scenario from the domain of logistics. A truck delivers a box that contains frozen peas to a warehouse. The box should be unloaded and stored in the cold store of the warehouse before the cold chain is broken. The unloading and storing are handled by a warehouse worker. Because the peas are

deep-frozen, it is important that they reach the cold store within a certain time frame when the unloading begins. This will be our running example for the rest of the document. Figure 1 shows an illustration of the local environment and all participants, figure 2 the necessary activities as basic flow. The truck transports the box to the warehouse. A worker from the warehouse is responsible to unload the box from the truck. Finally the box is moved to the cold store by a warehouse worker.

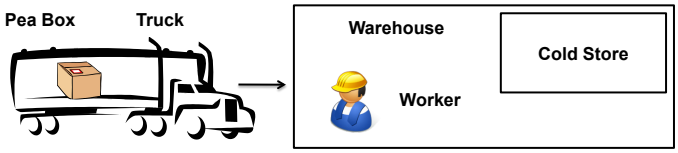

Fig. 1. Logistics Scenario overview

Fig. 2. The basic flow for storing the box

3 Related Work

We found two general approaches that enhance workflow modeling with context information: workflows with context integration [3–7] that allow to retrieve and react to context information and personal workflows [8–10] that are tailored to the actions of a single human user.

The context integration approaches provide modeling mechanisms that allow to specify a context-aware transition conditions between the activities. An early approach based on BPEL and WebServices was proposed by Han et al. [3]. Their *ubiquitous workflow description language (uWDL)* adds triples consisting of subject, verb and object to a given transition in a workflow. When the context is matched the transition is taken. uWDL has developed to a complete framework [4] including modeling tools and support of context aware services [5]. With *Context4BPEL* [6] Wieland et al. allow the workflow to directly access context information and take decisions on that information. They further

extend their system [7] to be able to process more complex queries and to facilitate a higher level of abstraction in context modeling. However uWDL as well as Context4BPEL rely on comprehensive modeling and provide no flexibility at modeling time, what kind of context information is relevant at runtime.

In the area of personal workflows Hwang et al. [8] proposed a proprietary workflow format for personal processes that provides a great deal of flexibility for the individual tasks. On the downside the modeling of the flow as well as monitoring its state of execution rely mostly on direct user input. The *sentient processes* [9] are also based on a proprietary language. Such a process is tailored to a single user that wants to execute a sequence of tasks in a pervasive environment. The later extension, *PerFlows* [10], allows the user to flexibly decide in which order he executes his tasks. They can be executed automatically, rolled back, skipped and are context aware to the behavior of the owner. However, the Personal Workflows as well as the PerFlows allow only the assignment of a single user to the flow. The user is static and the flow can only react on context information of that user or direct input. Both concepts do not consider additional context information that becomes relevant during execution.

PerCollab [11] envisions another approach for pervasive human interaction. The authors extend the BPEL with a new activity that allows transparent user interaction using heterogeneous communication channels ranging from desktop PCs to PDAs and even mobile phones. This presents a convenient way to integrate human interaction into a workflow, but as the interaction is handled transparently the flow is not aware of its environment and thus cannot react to it.

4 Situated Workflows

In order to relate our application to the physical world, we must first create a formal model of the physical context. In the next section, we introduce a context model that is built on object-oriented *entities*. Given this formal representation, we can attach a flow to an entity creating a direct relation to a physical object. This *Flow Attachment* is explained in section 4.2. Using the attachment, we define the relevant context information in the local environment of the flow. We call this concept a *Context Frame*. During the runtime of the flow we enforce application constraints on the actual context and use available context to control the flows behavior when certain events in the relevant environment happen. The constraint handling is implemented with *Context Constraints* and the event handling with *Context Events*.

4.1 Entity Model

The entity model is our formal representation of the real world. Every real-world artifact that is considered in our system is encapsulated by an *entity*. Each entity has an identifier and an *entity type*. For our application scenario, we consider the following three entity types; 'truck', 'box', 'worker'. The relevant instances

for our scenario are: 1. The box that contains the frozen peas. 2. The truck that transports boxes to the warehouse. 3. The worker who is able to carry boxes, unload them from the truck and put them to different storage places like the cold store.

The type of the entity further determines which properties and events the entity has. An *entity property* is a variable in the entity type that represents a real-world property of the corresponding physical object. Examples for properties are the temperature of the box, or the position of the worker. An *entity event* is generated when something happens at a certain point in time in the real world that causes a change for an entity. An example for this might be a worker that picks up a box which results in a pickedUp-Event for the box. We assume that the properties and events defined in the entity types can be sensed from the real world by appropriate context recognition systems that are integrated in the context access layer. For the sake of simplicity, the domain of values every attribute can have is known in advance. Whenever we refer to either an entity property or an entity event we also speak of an *entity attribute*. Entity types are organized in an entity type hierarchy and allow single-parent inheritance from other entity types similar to object-oriented programming.

4.2 Flow Attachment

Based on the entity model, we introduce the necessary extensions to the flow modeling language to create situated flows. We build our extensions using two properties of the flow modeling language. First, the flow modeling language provides scopes as a modeling element. A scope groups a subset of the activities in a workflow model employing visibility rules. Every information defined in a scope can only be accessed by the activities in the scope. We represent scopes as dotted rectangles around the activities. Secondly, the flow model is easily extendable with annotations that can be added to scopes.

Given these two tools, we attach a flow to an entity, by adding an attachment annotation to a scope. In order to improve the flexibility, attachment happens in two distinct phases. In the first phase, during modeling time of the flow, the modeler adds the attachment annotation to a scope. Only the entity type of the attachment is defined at this point. In the second phase, when the flow is running the attachment is resolved to an entity, that is an instance of the entity type specified in the attachment. Then the flow is actually attached to that entity. Figure 3 shows an attachment of the box flow to the box. The small dotted rectangle (A) represents the attachment annotation, its code depicted in the box. The actual box instance is resolved at runtime. A single scope can have multiple attachments annotated, which allows a flow to be attached to multiple entities. The entity type of the attachment is static during runtime but the attached entity can be exchanged under certain conditions. For example, a flow could be attached to a worker and this worker finishes his shift while the flow is still running. Because the worker is unavailable, the flow needs to resolve another worker, to continue execution.

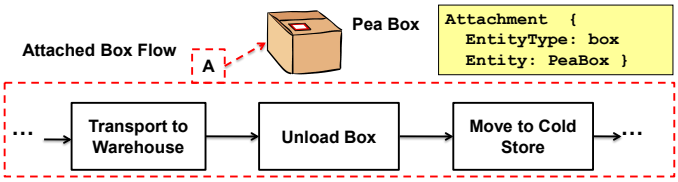

Fig. 3. Extension of the basic workflow model to a situated workflow

When a flow is attached to an entity the flow becomes situated. It is logically co-located with the attached entity and the entity becomes a relevant part of the local environment the flow can respond to. It is also likely that the flow will access some of the context information of the attached entity. Because of this the context access layer can dynamically provide efficient access to the context information of attached entities to the flow.

4.3 Context Frames

The environment of the flow might not only include attached entities but also other entities which are not known at design time. Those entities become relevant to the flow during its runtime because they have some contextual relation to the actual entity the flow is attached to. The modeler can add a *Context Frame* to the flow providing access to more entities in the flows local environment based on that contextual relationship. In our scenario, there is a worker responsible for unloading the box from the truck. But flow modeler cannot know the actual context of each worker in the warehouse. So the flow modeler adds a context frame that defines those workers as relevant that can assist the unloading and storing process. In Figure 4, the box flow is depicted including a Context Frame that defines the relevant workers for the flow. We add the Context Frame as new modeling element to the flow modeling language. Similar to the Flow Attachment, the Context Frame is annotated to a scope in the flow model.

A Context Frame defines any number of so called *entity sets* that contain the entities that are relevant to the flow. The modeler can define an entity set either combining two existing ones using set operations (union, intersection, set theoretic difference) or create new entity sets using certain *Filters*.

We propose the use of three different filters. The first one is a *Type Filter* that restricts the type of entities that can be in the set. The type filter in our example is set to worker so that only workers can be an element of the set and not other entities. The second one is an *Attribute Filter*. It lists the number of entity attributes that are considered relevant and only these attributes are updated by the context system. We apply this filter to further reduce the amount of context information that must be provided to the flow by the context access layer. The third filter is the so called *Range Filter*. The range filter removes the entities, whose attributes are not in range of a defined domain of values. It consists

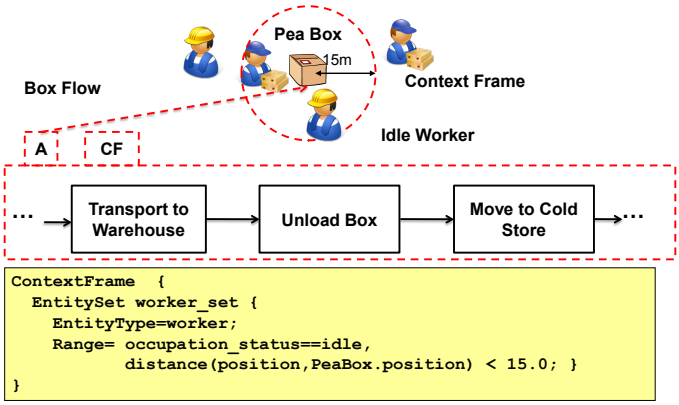

Fig. 4. Context frame that covers all free workers within 15m of the box

of a number of expressions and an entity must fulfill all these expressions in order to be a member of the entity set. In our example, we use two range filter expressions. As the box flow in Figure 4 is only interested in workers which are idle, occupied workers are filtered out. The second range filter expression removes all workers from the set that are not close enough to be relevant for further execution. The Context Frame can access the context information of the attached entities. The information is used to define the relation between the entities of the set and the attached entity. This way the content of the entity set is directly dependent on the actual context of the box. Because of the changing context of the attached entity, the actual elements of the entity set are calculated dynamically at runtime taking the relation into account. When a Range Filter is defined without a relation to an attached entity we call the enclosing Context Frame an absolute Context Frame. Otherwise it is a relative Context Frame. The context information in a Context Frame can be accessed, when the flow executes activities in the scope.

4.4 Context Constraints and Events

Context Constraints and *Context Events* are two mechanisms to utilize the context information provided from attachment and context frames. Both have a similar structure but different evaluation semantics. We first describe the common structure and then discuss the different semantics using the application scenario. Both, Context Constraints and Context Events, consist of a *context condition* and a *handler*. The context condition monitors the situation and invokes the handler when necessary. The condition itself consists also of two separate parts. A logical *predicate* in first order logic and an *event qualifier*. The predicate is defined over the entity properties from the attached entities and from the context frames. The evaluation rules of the context condition are controlled by the event qualifier. We provide two different event qualifiers. The

onEvent qualifier evaluates the condition when a certain entity event (c.f. 4.1) is triggered. The *betweenEvents* qualifier evaluates the condition continuously during the specified events. The event qualifiers are defined on the available entity events. We also provide two special events to allow evaluation when a scope is entered or left. These two events are the onEntry-Event and the onExit-Event.

Figure 5 shows the further extended box flow. We have annotated a Context Constraint (CC) and a Context Event (CE) to the inner scope that encloses the unloading task. The semantics for the Context Event are defined as follows. When the context condition of a Context Event is evaluated and the predicate is true, then the handler is invoked. When the truck has reached the warehouse, the flow waits until a worker is close enough to the box. When the entity set in the Context Frame becomes non empty the handler is invoked. The flow gets a notification and can continue its execution, because a worker is available that can start the unloading task. The semantics for the Context Constraints are basically inverted. When the predicate evaluates to false the handler is invoked. While the box is being unloaded, the condition of the CC is evaluated, because the flow gets notified about the peaBox.pickedUp event. In our example scenario we consider a context constraint that monitors if the temperature of the box is always below -18 degrees Celsius so that the cold chain is maintained. If the predicate becomes false at any point in time until the task is completed the handler of the Context Constraint is invoked. The handler then tries to resolve the constraint violation. Additional information on integrated constraint handling in flows can be found in our previous work [12].

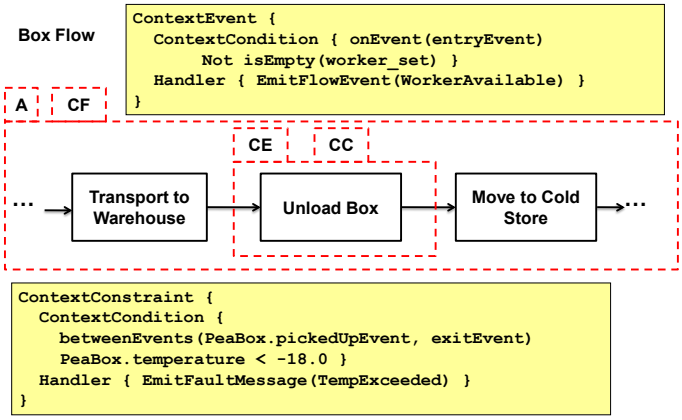

Fig. 5. The box flow with a Context Event and a Context Constraint

4.5 Example

To demonstrate the effects of our extensions we show a walkthrough of our scenario pointing at the effects of our new modeling elements. The box flow handles the transport process of the flow. When the flow is invoked it is first attached to the pea box. The box is loaded on a refrigerated truck and transported to the warehouse where it should get stored for some time. On arrival of the truck the box flow waits until it gets notified that a worker has - literally speaking - entered the Context Frame, i.e. fulfills the range filter of the Context Frame. The worker gets notified about his next task and starts to unload the box from the truck into the warehouse. When the box flow receives the pickedUp event from the peaBox the Context Constraint starts monitoring the temperature of the box. Note that the flow only knows and can subscribe to the pickedUp event and the temperature values because he is attached to the box. Without the attachment from this flow there might be no other application that is interested in context information about that box and the context access layer would not provide them. If the worker carries the box to the cold store in time the flow will continue normally. However, should the temperature rise above the value in the constraint, the system reacts to the change. The flow might be adapted to the new situation and activities could be added e.g. an additional quality check on the goods. But the constraint violation could also lead to the termination of this flow and trigger the removal of the box from the usual delivery.

5 Discussion

The extensions we provided for the flow modeling language enable a flow modeler to create situated workflows. He can describe the local environment of the flow based on Flow Attachment and Context Frames. The modeler can further apply our constraint and event handling mechanisms which allow the flow to adapt to the environment. The main advantage of our approach is the reduced amount of context information that is relevant and the fact that relevant context information can be determined at runtime. The context provisioning system only has to provide and maintain information that are relevant to the flow applications (i.e. attached entities and entities defined in Context Frames). The flow benefits from a much smaller range of context in its local environment it needs to monitor and adapt to. But our approach leaves some open challenges. Our approach relies on quite comprehensive knowledge about context information. Every context that is modeled in the flow should be available during runtime. Another aspect is the quite expensive dynamic calculation of the visible context for a single flow. We will investigate the use of aggregation techniques for entity sets based on the assumption that similar flows will have similar requirements on context information.

6 Conclusions and Future Work

In this paper we introduced a novel way to model context aware workflows. Our approach provides more flexibility in modeling than the existing approaches. We dynamically restrict the amount of context information so that the flow is only aware of relevant context. We further provided a constraint and event handling mechanism on top of our context modeling approach that allows to monitor the execution and to trigger adaptation. As we discussed above the provisioning of the entity sets may be rather expensive. We will investigate how to optimize the calculation of entity sets in a system that hosts a large number of situated workflows. This includes the adaptation of the context provisioning system to the needs of a single or similar situated flows.

References

1. Hull, R., Neaves, P., Bedford-Roberts, J.: Towards situated computing. In: ISWC '97: Proceedings of the 1st IEEE International Symposium on Wearable Computers, Washington, DC, USA, IEEE Computer Society (1997) 146
2. Herrmann, K., Rothermel, K., Kortuem, G., Dulay, N.: Adaptable pervasive flows - an emerging technology for pervasive adaptation. In Society, I.C., ed.: Proceedings of the SASO 2008 Workshops. (2008)
3. Han, J., Cho, Y., Choi, J.: Context-aware workflow language based on web services for ubiquitous computing. In Gervasi, O., Gavrilova, M.L., Kumar, V., Laganà, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K., eds.: ICCSA (2). Volume 3481 of Lecture Notes in Computer Science., Springer (2005) 1008–1017
4. Han, J., Cho, Y., Kim, E., Choi, J.: A ubiquitous workflow service framework. In Gavrilova, M.L., Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganà, A., Mun, Y., Choo, H., eds.: ICCSA (4). Volume 3983 of Lecture Notes in Computer Science., Springer (2006) 30–39
5. Shin, K., Cho, Y., Choi, J., Yoo, C.W.: A workflow language for context-aware services. In: MUE '07: Proceedings of the International Conference on Multimedia and Ubiquitous Engineering, IEEE Computer Society (April 2007) 1227–1232
6. 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 Vol.2, Trondheim, Norway, June 11-15th, 2007, Tapir Acasemic Press (Juni 2007)
7. Wieland, M., Kaczmarczyk, P., Nicklas, D.: Context integration for smart workflows. In: Proceedings of the Sixth Annual IEEE International Conference on Pervasive Computing and Communications, Hong Kong, IEEE computer society (März 2008) 239–242
8. Hwang, S.Y., Chen, Y.F.: Personal workflows: Modeling and management. In: Lecture Notes in Computer Science. Volume 2574/2003 of LNCS., Springer (2003) 141–152
9. Urbanski, S., Becker, C., Rothermel, K.: Sentient processes - process-based applications in pervasive computing. In: PerCom Workshops, IEEE Computer Society (2006) 608–611
10. Urbanski, S., Huber, E., Wieland, M., Leymann, F., Nicklas, D.: Perflows for the computers of the 21st century. PerCom Workshops **0** (2009) 1–6

11. Chakraborty, D., Lei, H.: Pervasive enablement of business processes. In: PerCom, IEEE Computer Society (2004) 87–100
12. Eberle, H., Föll, S., Herrmann, K., Leymann, F., Marconi, A., Unger, T., Wolf, H.: Enforcement from the Inside: Improving Quality of Business in Process Management. In: 2009 IEEE International Conference on Web Services (ICWS 2009), Los Angeles, IEEE Computer Society (Juli 2009)