



**The 1st International Workshop on
Process Management for Highly Dynamic and
Pervasive Scenarios
(PM4HDPS)**

Milan, Italy, 1 September 2008

**In conjunction with
the 6th International Conference on
Business Process Management (BPM'08)
Milan, Italy, 1-4 September 2008**

Workshop Proceedings

Massimiliano de Leoni

Dipartimento di Informatica e Sistemistica
SAPIENZA – Università di Roma
deleoni@dis.uniroma1.it

Schahram Dustdar

Distributed Systems Group
Vienna University of Technology
dustdar@infosys.tuwien.ac.at

Arthur H.M. ter Hofstede

Faculty of Information Technology
Queensland University of Technology
arthur@yawlfoundation.org

Hypergraph-based Modeling of Ad-Hoc Business Processes

Artem Polyvyanyy and Mathias Weske

Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
D-14482 Potsdam, Germany
{Artem.Polyvyanyy,Mathias.Weske}@hpi.uni-potsdam.de

Abstract. Process models are usually depicted as directed graphs, with nodes representing activities and directed edges control flow. While structured processes with pre-defined control flow have been studied in detail, flexible processes including ad-hoc activities need further investigation. This paper presents flexible process graph, a novel approach to model processes in the context of dynamic environment and adaptive process participants' behavior. The approach allows defining execution constraints, which are more restrictive than traditional ad-hoc processes and less restrictive than traditional control flow, thereby balancing structured control flow with unstructured ad-hoc activities. Flexible process graph focuses on what can be done to perform a process. Process participants' routing decisions are based on the current process state. As a formal grounding, the approach uses hypergraphs, where each edge can associate any number of nodes. Hypergraphs are used to define execution semantics of processes formally. We provide a process scenario to motivate and illustrate the approach.

Key words: business process modeling, hypergraph-structured process, ad-hoc process, process formalism, execution semantics

1 Introduction

Process models provide companies efficient means for managing their daily routines. A business process model consists of a set of activity models and execution constraints between them. A business process instance represents a concrete case in the operational business of a company, consisting of activity instances [21]. A business process model represents a collection of process instances that handle the same business task. The key idea is that a business task can be managed differently under special conditions, thus resulting in different process instances. Process models allow flexible business task handling where a process instance can evolve multiple possible scenarios. Process model reasonable lifecycle, in static environment, usually assumes large level of flexibility at early stages slowly evolving to best-practice scenarios at its maturity.

For non-trivial processes, characterized mainly by dynamic environment, the level of required flexibility for its participants is usually high and should be

kept at all stages of process lifecycle. Such processes can be observed in complex adaptive systems, also made up of people or artificial intelligence agents. Process flexibility is no longer solely explained by the variability in business task scenarios but also by variability in participants’ (process executors’) behavior and ongoing environment changes. These systems are dynamic and open, rather than simple and optimized for best-practice scenarios. A good example of such a system is an educational service system where a student can undertake different paths to acquire a service of education. At the same time, other system participants are also flexible in their behavior and execute process tasks in the ad-hoc manner. For highly dynamic environments many research issues need to be addressed in a different way, e.g., process modeling.

In this paper we present a formal approach for modeling flexible business processes driven by adaptive behavior of process participants—flexible process graph (FPG). Process routing mechanism is enhanced to allow participant adaptation to environment change on a control flow level by allowing a flexible choice of the next activity to accomplish at each state of a process instance. In the core of the idea lies generalization of a process graph structure to a hypergraph structure. Further, the process execution semantics is defined for a hypergraph-structured process. A graph edge evolves from specifying a sequence control flow pattern on two activities to a set of activities that can be accomplished in the order determined when executing a process instance.

The rest of the paper is organized as follows. In the next section we motivate the proposed approach by investigating modeling and execution support for processes that represent large collections of process instances. The Business Process Modeling Notation (BPMN) [13, 14] ad-hoc process is taken as a starting point. In section 3 we introduce FPG. We present the core idea, formalism, execution semantics, and graphical representation. In section 4 we return back to the motivation scenario and apply FPG formalism to model the process. Related work is investigated in section 5. The paper closes with conclusions that summarize our findings.

2 Motivation

Processes governed by adaptive participants’ behavior caused by dynamically changing environment are characterized by large amounts of process instance variants. The problem of state of the art process modeling techniques for such scenarios can be identified as an attempt to model what should be done by a process participant. Adaptive participants’ behavior requires an extensive choice of next steps at each process state leading to a necessity of modeling constructs for each possible proceeding. As a motivation, we propose to take a look at a process that assumes large amount of process instances.

Figure 1 shows an example of the ad-hoc process from [13] that proposes to look at process of writing a book chapter. The example ad-hoc process includes six tasks: “*researching the topic*”, “*writing text*”, “*editing text*”, “*generating graphics*”, “*including graphics in the text*”, “*organizing references*”, and is

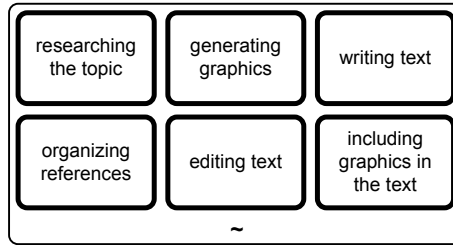


Fig. 1. BPMN ad-hoc process example

visualized using BPMN 1.0 notation. Let us assume a simplified version of the process when the ad-hoc process is configured for sequential execution and each task has to be accomplished exactly once. Under the assumption the model from Figure 1 describes an extreme case of $6! = 720$ possible process instances.

One can introduce additional dependencies between tasks in the process, such as “writing text” must be executed before “editing text”: the inverse order is just not realistic. Let us identify a complete set of constraints. Let the desired process be such that the task of “researching the topic” should always happen first. Further, the task of “including graphics in the text” can only be performed once both “writing text” and “generating graphics” tasks are accomplished. Finally, it

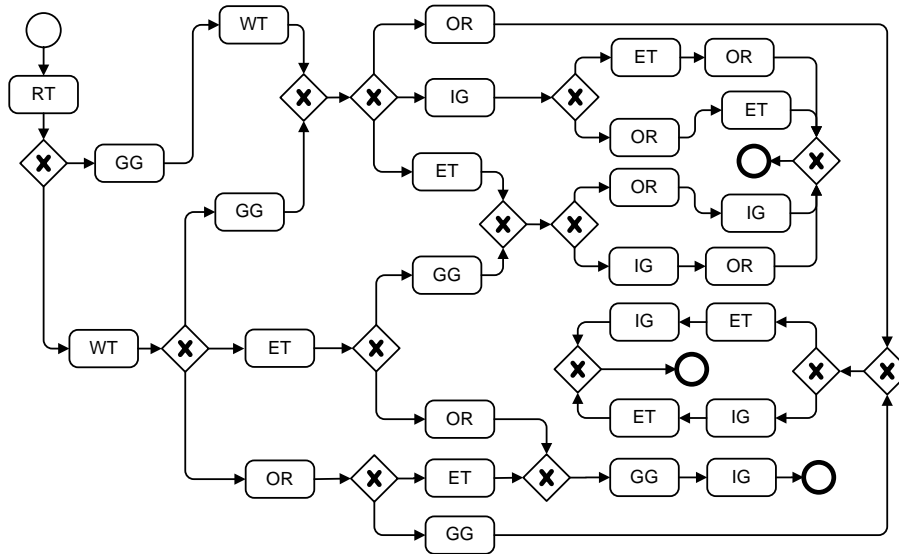


Fig. 2. BPMN process model that covers concretized scenario of 18 instances. RT=“researching the topic”, WT=“writing text”, ET=“editing text”, GG=“generating graphics”, IG=“including graphics in the text”, and OR=“organizing references”

is possible to proceed with “*editing text*” or “*organizing references*” once finished with “*writing text*”. Described constraints reduce the amount of desired model instances from 720 to 18.

BPMN ad-hoc processes are not executable, but rather are just requests to a process participant on a fulfillment of work packets, e.g., to write a book chapter, containing suggestion on possible work packets decomposition to smaller sub-tasks. In order to provide execution support for the concretized version of the sample process, a model has to formally represent all the execution constraints.

Figure 2 shows the BPMN diagram to model the exact collection of desired process instances. The model proposed is one possible solution. It is feasible to derive other models that describe exactly the same process by trading between the number of gateways and activity model occurrences in the diagram.

The model from Figure 2 represents another extreme case of a precise process specification suitable for execution. Besides that it is a complex task to derive such a model, the model suffers of elements explosion, i.e., complex gateway structures and multiple activity model occurrences. Moreover, any model modification becomes laborious, e.g., you may want to introduce a review task between “*writing text*” and “*editing text*” tasks. Local model modifications are not enough to reflect global ad-hoc process logic. The model should be adapted to incorporate global execution constraints, i.e., constraints of modified activities with all other activities of the model.

In the next section we introduce FPG—a formal approach that allows representing large collections of process instances. Afterwards, we will return back to the concretized process scenario and implement it as FPG.

3 Flexible Process Graph

Prior to start with presentation of the flexible process graph approach, we briefly summarize what is desired to achieve. FPG is envisioned as a technique for modeling processes that incorporate adaptive participants’ behavior on a control flow level and allows specification of task execution constraints. The modeling technique has to permit intuitive process visualization. Developed process models should represent large collections of process instances in a compact way. Finally, FPG has to provide a formal process execution semantics that allows tracking of the process execution state.

In the core of the idea for modeling flexible process graphs lies generalization of a directed graph edge which defines a sequential execution on connected activities. Figure 3(a) shows a directed graph edge that specifies a sequence control flow pattern. An application of the sequence control flow pattern will result in activity *b* to get enabled (become available for execution) only after activity *a* has terminated (was executed). Thus, a process participant is dictated on what should be done next in a process instance.

Figure 3(b) gives an example of a hyperedge (generalization of a graph edge) that connects 3 nodes: *a*, *b*, and *c*. As opposite to a graph-based sequence control flow pattern, one can allow that within a hyperedge a process participant

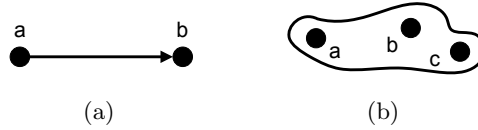


Fig. 3. (a) Graph-based control flow edge, (b) hypergraph-based control flow edge

can choose which activity to execute next. The decision reflects his/her adaptive behavior in the context of a dynamic environment. A process model becomes hypergraph-, rather than graph-structured. On the first step of the process execution either activity a , b , or c can be performed. On the next step selection is reduced to 2 non-terminated activities. The third step is completely deterministic and finishes execution of the edge.

3.1 Hypergraph Preliminaries

In mathematics, a hypergraph is generalization of a graph [10, 11]. Hypergraph edges (hyperedges) are arbitrary sets of nodes. Thus, a hyperedge is an edge that connects multiple nodes.

Formally, a hypergraph is a pair (N, E) where N is a set of elements, called nodes or vertices, and E is a set of non-empty subsets of N , called hyperedges. Therefore, E is a subset of $\mathcal{P}(N) \setminus \{\emptyset\}$, where $\mathcal{P}(N)$ is the power set of N . Figure 4 visualizes the example of a hypergraph. Here, $N = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, $E = \{e_1, e_2, e_3, e_4\}$, $e_1 = \{v_1\}$, $e_2 = \{v_2, v_3, v_4, v_5, v_6\}$, $e_3 = \{v_4, v_5, v_6\}$, and $e_4 = \{v_5, v_8\}$.

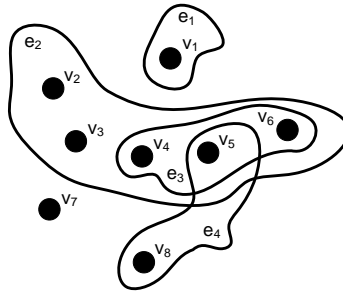


Fig. 4. Hypergraph example

Hypergraph structure is used as the structure of flexible process graphs. Hyperedges in a hypergraph can intersect. This fact is used to derive semantics for passing control flow initiative from a hyperedge to a hyperedge, thus specifying process instance development.

3.2 Formalism

In this section we present the flexible process graph formalism which includes definition of process state and process execution semantics. Process execution semantics describes process state transition principles.

Let us start with the definition of a flexible process graph.

Definition 1. A flexible process graph (FPG) is a triple (A, E, T) where:

- A is a finite set of activity nodes
- E is a finite set of edges $e = \langle I(e), O(e) \rangle \in E$, $A \cap E = \emptyset$
 - $I : E \rightarrow \mathcal{P}(N)$ is a function defining edge input nodes
 - $O : E \rightarrow \mathcal{P}(N) \setminus \emptyset$ is a function defining edge output nodes
 - $\forall e \in E : I(e) \cap O(e) = \emptyset$
- T is an edge type function, $T : E \rightarrow \{\text{and}, \text{xor}, \text{or}\}$.

In the case of FPG, a process structure is given by a hypergraph (A, E) . Activities A are modeled as hypergraph nodes. Each edge in FPG is a subset of activity nodes. A hyperedge e is split into two subsets of input ($I(e)$) and output ($O(e)$) nodes to obtain a *directed* hypergraph. The subsets are mutually disjoint. Edge outputs, and thus edges, can not be empty. Unlike regular graph-structured process models that contain special routing nodes—gateways, FPG introduces edge types that implement routing decisions.

Directed edges in FPG are crucial for definition of the process execution semantics. A regular process graph directed edge defines sequence control flow pattern; an activity in a process is enabled for execution after the completion of a preceding activity. Similar, output activities in an FPG edge are enabled for execution after the completion of all of the edge input activities. Once enabled, the edge output activities can be accomplished in any order. If an FPG edge has no input activities, output activities of such an edge are enabled for execution at process instantiation. These activities are the first candidates proposed for completion to process participants.

FPG gives a compact way of representing large collections of process instances as the main building block of a process, hyperedge, represents a complete set of process instances on contained nodes. Adaptive participants' behavior in a condition of dynamic process environment is achieved with the help of edge routing decisions and further selection of an activity to accomplish next from the edge output set. A process participant is guided while process instance execution by edge routing decisions (also taken based on a dynamic process environment state) and exposes adaptive behavior by determining the order in which to accomplish proposed activities. The order in which activities are performed is adapted by the process participant at execution time as a reaction to environment changes.

3.3 Execution Semantics

FPG execution semantics defines allowed process instances. The structure of FPG is fixed and does not change during execution of a process instance. Dy-

namics of a process represented as FPG is specified by process state transitions. A state of FPG may change according to state transition rules.

Definition 2. A state of a flexible process graph (A, E, T) is defined by a state function $S : A \rightarrow \mathbb{N}_0 \times \mathbb{N}_0$ mapping a set of activity nodes onto the pairs of natural numbers including zero ($\mathbb{N}_0 = \mathbb{N} \cup \{0\}$).

When in certain state, each activity node a of FPG is assigned two numbers $S(a) = (\omega, \beta) \in \mathbb{N}_0 \times \mathbb{N}_0$. $S_\omega(a) = \omega$ (*white* tokens) specifies the number of instances of activity a that need to be accomplished from now on in the process instance. Respectively, $S_\beta(a) = \beta$ (*black* tokens) specifies the number of activity instances so far accomplished in the process instance. The FPG state function role is similar to the marking (or state) function of Petri nets [15].

Process Instantiation Upon creation of a new process instance the FPG state function has to be initialized. Initialization is performed in two steps:

1. $S(a)$ is set to $(0, 0)$ for all $a \in A$
2. For each activity $a \in A$ the initial enabling is performed.

The initial activity enabling identifies a set of start activity candidates. An activity a is enabled at process start if $\epsilon^*(a)$ holds, where ϵ^* is a predicate that specifies initial activity enabling condition $\epsilon^* : A \rightarrow \{true, false\}$.

$$\epsilon^*(a) = \exists e \in E : a \in O(e) \wedge I(e) = \emptyset \wedge cond(e, a)$$

The $cond : E \times A \rightarrow \{true, false\}$ predicate implements edge type $t \in T$ routing decisions (e.g., $\forall a \in O(e) : cond(e, a) = true$, if $T(e) = and$). If $\epsilon^*(a)$ holds, the process state S is modified to give S' , such that $S'(a) = S(a) + (1, 0)$.

Activity Firing An activity a can fire in an FPG process instance if it is enabled ($S_\omega(a) > 0$). Activity firing results in the process state S change to S' , such that $S'(a) = S(a) + (-1, 1)$, i.e., one white token gets painted black. Activity firing is instantaneous, consumes no time, and indicates a completion of the corresponding activity. After activity a has fired, the activity enabling has to be performed on a set composed of output activities of a :

$$\bigcup_{\{e \in E | a \in I(e)\}} O(e)$$

Activity Enabling Activity node enabling defines rules for modification of activity node state and thus of process instance state. An activity a can be enabled after execution of an activity a_β if $\epsilon(a_\beta, a)$ holds, where ϵ is a predicate that specifies activity enabling condition $\epsilon : A \times A \rightarrow \{true, false\}$.

$$\epsilon(a_\beta, a) = \exists e \in E \forall a_i \in I(e) : a_\beta \in I(e) \wedge a \in O(e) \wedge S_\beta(a_i) \geq S_\beta(a_\beta) \wedge cond(e, a)$$

Activity a enabling depends on execution of the preceding activity, e.g., a_β . Activity a can be enabled if there exists an edge e such that a is the output activity of e and a_β is the input activity of e . Further, for each input activity a_i of the edge e it holds that the number of accomplished instances of a_i is at least the number of accomplished instances of a_β , i.e., the number of black tokens of each of the input activity of the edge e is at least the number of black tokens of the activity a_β . Finally, the edge e type $t \in T$ condition must hold.

If $\epsilon(a_\beta, a)$ holds, the process state S is modified to result in state S' , such that $S'(a) = S(a) + (1, 0)$.

Process Termination A process instance terminates when there is no activity to execute, i.e., no activity is enabled, $\forall a \in A : S_\omega(a) = 0$.

3.4 Graphical Representation

Graphical representation of FPG can be mapped onto triple (A, E, T) and vice versa (in both directions). For instance, Figure 5 visualizes the example of a flexible process graph applying developed graphical notation. Developed notation preserves hypergraph visualization approach proposed in Figure 4 and differentiates presentation of edge types and activities (input activities are located on the borderline of the corresponding edge region, e.g., node a from edge e_2).

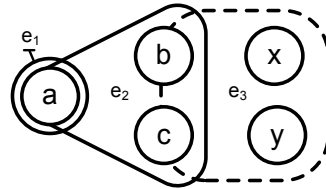


Fig. 5. Flexible process graph example

The model from Figure 5 can be mapped onto (A, E, T) triple such that: $A = \{a, b, c, x, y\}$, $E = \{e_1, e_2, e_3\}$, where $e_1 = \langle \emptyset, \{a\} \rangle$, $e_2 = \langle \{a\}, \{b, c\} \rangle$, and $e_3 = \langle \{b, c\}, \{x, y\} \rangle$. Finally, $T(e_1) = T(e_2) = \text{and}$, $T(e_3) = \text{xor}$. Here, we use circles to represent process activity nodes.

Apparently, one can follow the reverse direction and visualize FPG by possessing formal process definition in a form of (A, E, T) triple.

4 Flexible Process Graph Example

After having presented the flexible process graph approach for modeling processes that allow adaptive behavior in the context of ongoing environment changes, we would like to return back to the example process scenario from section 2. We will now formalize its concretized version as FPG.

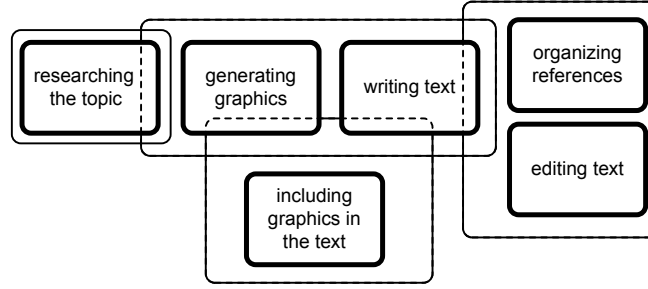


Fig. 6. FPG model of process from Figure 2

Figure 6 visualizes the flexible process graph that is obtained from triple (A, E, T) , where A consists of RT = “researching the topic”, WT = “writing text”, ET = “editing text”, GG = “generating graphics”, IG = “including graphics in the text”, and OR = “organizing references”, $E = \{e_1, e_2, e_3, e_4\}$, $e_1 = \langle \emptyset, \{RT\} \rangle$, $e_2 = \langle \{RT\}, \{GG, WT\} \rangle$, $e_3 = \langle \{GG, WT\}, \{IG\} \rangle$, and $e_4 = \langle \{WT\}, \{OR, ET\} \rangle$, and function T is such that $T(e_1) = T(e_2) = T(e_3) = T(e_4) = \text{and}$.

Proposed FPG formal model represents the same process as in Figure 2 in a compact way. Execution semantics of the FPG model allows the exact 18 process instances. First, upon process instantiation, white and black tokens for each activity node are set to 0. After the process initialization phase, the activity RT gets enabled, $S(RT) = (1, 0)$. Because it is the only enabled activity, it will eventually fire and result in $S(RT) = (0, 1)$. After activity firing, activity enabling takes place on the set of output activities $\{GG, WT\}$. As a result, $S(GG) = S(WT) = (1, 0)$. The firing of the WT activity enables OR and ET . Finally, after the second activity from the set fires and $S(GG) = S(WT) = (0, 1)$, the activity IG gets enabled. The process continues until $S = (0, 1)$ for all the process activities. Then, the termination condition holds and process terminates.

The FPG model allows adaptive process participants’ behavior forced by environment changes. E.g., once finished with “researching the topic” the process participant can proceed with “generating graphics” or “writing text”, but might be constrained to the option of “writing text” because of the current environment that provides no access to a graphics editor tool.

5 Related Work

Process models that allow adaptation to environment changes cover large collections of process instances. Expansion of process allowed instances in models can be achieved by techniques employing adaptive changes in process models. Three known dimensions of change in process models are dynamism, adaptability, and flexibility [19]. Following, we discuss approaches that address the dimensions of change in process models.

ADEPT_{flex} [16] is a formal foundation for support of dynamic structural changes of running process instances. First, a formal workflow model is defined (ADEPT), then a complete and minimal set of change operations (ADEPT_{flex}) is introduced. Additionally, correctness properties are defined to check whether a change can be applied to a process instance.

Similar, in [20] the ability of workflows to adapt the structure of running instances is addressed. The approach is based on a formal model of workflow schema. A workflow schema is a directed graph structure with homogeneous nodes.

Worklets [3, 4] introduce dynamism in workflow through the support of flexible work practices. Worklet is an extensible repertoire of sub-processes assigned to each workflow task. When executing a process instance, a dynamic selection of appropriate sub-process from available is performed.

In [19] the authors introduce the notion of open workflow instance. An open instance consists of a core process and several pockets of flexibility. A pocket of flexibility is a set of workflow fragments and a special build activity. The build activity provides rules for concretizing the pocket with a valid composition of available fragments at execution time.

All of the presented approaches extend amount of model initially allowed process instances by performing process graph transformations. The primary goal of these approaches is to address variability, exceptions handling, and change in business case handling. By introducing model transformation rules, the initial model becomes non-transparent. In the case of process participant adaptive behavior it is extremely hard to foresee and manage all of the possible model changes, to introduce new ones, or to remove already developed. Contrary to the discussed approaches, the developed approach of FPG is a modeling technique that allows models to naturally represent adaptive participants' behavior in the context of environment changes.

The applicability of hypergraphs for modeling task was studied in [9]. Inspired by hypergraph formalism, the authors propose metagraph structure. As stated in [5], metagraphs were not initially intended for modeling workflows. The primary goal of metagraphs is to represent transformation relations between two sets of objects. Later, the approach was extended to represent workflows [6, 7, 8]. Activities in a metagraph-based workflow are represented by arcs that relate objects consumed and produced during activity execution.

Case handling [1, 2, 12, 17] is a paradigm for support of flexible business processes. It is strongly based on data as the typical product of the processes. In workflow management what should be done in a process instance is determined by predefined control structures. Case handling focuses on what can be done to achieve a business goal. In case handling, process participants decide on how to approach a goal state.

Similar to metagraphs, FPG employs hypergraphs for the task of workflow modeling. However, FPG follows the well-accepted paradigm of modeling activities as graph nodes, rather than as arcs. The generalized graph structure, which is hypergraph, provides more space for representing workflow instances

that assume adaptive participants' behavior in the condition of highly dynamic execution environment. Further, similar to case handling, a process participant is allowed to decide what needs to be done to achieve a process goal. But unlike in the case handling case, the decision is taken based on the activities already performed (process history), and not based on data objects at hand.

6 Conclusions

This paper introduced flexible process graph as a new paradigm for supporting modeling and execution of flexible business processes driven by adaptive participants' behavior in the context of environment changes. We started out this survey with a motivating example of a process scenario. The example shows the lack of support for modeling and executing the processes that do not fit into the paradigm of modeling strict sequential control flow constructs. We then proposed FPG formalism, as the mean for solving identified problems of present-day workflow modeling techniques. The FPG process execution semantics, including process instantiation, activity enabling, activity firing, and process termination condition, was provided. The graphical notation for visualizing FPG processes, which is based on proposed formalism, was presented.

Flexible operational processes where participants can undertake different paths to reach a goal state are characterized by large amounts of process instance variants. If one attempts to derive a model that captures all the variants to allow process execution support, the model has to be specified to represent all the constraints. FPG allows describing large amounts of activity execution constraints in a compact way. Additionally, FPG provides process execution support. The drawback of FPG is somewhat less intuitive process visualization, as compared to well-accepted graph-based notations. FPG is not aiming at replacement of existing process modeling techniques, but at complementing them, e.g., FPG can be applied in the region of a process where extensive flexibility is required.

FPG allows easy process modification to capture new requirements. In the case it is desired to introduce a review task between “*writing text*” and “*editing text*” in the concretized process scenario from section 2, minor modifications are required, i.e., to replace “*editing text*” task with a new “*make review*” task that once accomplished enables “*editing text*”. Such local model modifications change activity execution constraints globally following the FPG execution semantics.

Besides modeling of flexible processes, FPG allows representing basic control flow patterns: sequence, parallel split, synchronization, exclusive choice, simple merge, as well as arbitrary cycles and structured loops [18]. Investigation of the FPG applicability for modeling advanced control flow patterns, e.g., multiple instance patterns, cancelation patterns, or advanced branching and synchronization patterns, is the future work. FPG, as proposed in this paper, has no notion of time. Activities do not take time. Similar to Petri nets, FPG only signals for concurrent activity enabling. Introduction of true parallelism, i.e., formalization of simultaneous activity execution by different participants, is the next step.

References

1. W.M.P. van der Aalst and P. Berens. Beyond Workflow Management: Product-Driven Case Handling, 2001.
2. W.M.P. van der Aalst, M. Weske, and D. Grunbauer. Case Handling: A New Paradigm for Business Process Support, 2005.
3. Michael Adams, A.H.M. ter Hofstede, David Edmond, and W.M.P. van der Aalst. Implementing Dynamic Flexibility in Workflows using Worklets, 2006.
4. Michael Adams, A.H.M. ter Hofstede, David Edmond, and W.M.P. van der Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 4275 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2006.
5. Amit Basu and Robert Blanning. Metagraphs: A Tool for Modeling Decision Support Systems. *Manage. Sci.*, 40(12):1579–1600, 1994.
6. Amit Basu and Robert Blanning. Metagraphs in Workflow Support Systems. *Decis. Support Syst.*, 25(3):199–208, 1999.
7. Amit Basu and Robert Blanning. A Formal Approach to Workflow Analysis. *Info. Sys. Research*, 11(1):17–36, 2000.
8. Amit Basu and Robert Blanning. Workflow Analysis using Attributed Metagraphs. In *HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9*, page 9040, Washington, DC, USA, 2001. IEEE Computer Society.
9. Amit Basu and Robert Blanning. *Metagraphs and Their Applications (Integrated Series in Information Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
10. C. Berge. *Graphs and Hypergraphs*. Elsevier Science Ltd., 1985.
11. C. Berge. *Hypergraphs: The Theory of Finite Sets*. Amsterdam, Netherlands: North-Holland, 1989.
12. Christian W. Günther and W.M.P. van der Aalst. Modeling the Case Handling Principles with Colored Petri Nets.
13. OMG. *Business Process Modeling Notation Specification*, 1.0 edition, February 2006.
14. OMG. *Business Process Modeling Notation Specification*, 1.1 edition, January 2008.
15. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, Bonn, Germany, 1962. (In German).
16. Manfred Reichert and Peter Dadam. ADEPT flex—Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
17. H.A. Reijers, J. Rigter, and W.M.P. van der Aalst. The Case Handling Case, 2003.
18. Nick Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and Natalya Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical report, BPMcenter.org, 2006.
19. S.W. Sadiq, Wasim Sadiq, and M.E. Orlowska. Pockets of Flexibility in Workflow Specification. In *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling*, pages 513–526, London, UK, 2001. Springer-Verlag.
20. M. Weske. Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In *HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences-Volume 7*, page 7051, Washington, DC, USA, 2001. IEEE Computer Society.
21. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer Verlag, 2007.

Towards Process Models for Disaster Response

Dirk Fahland^{1*} and Heiko Woith²

¹ Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6, 10099 Berlin, Germany, fahland@informatik.hu-berlin.de

² GeoForschungsZentrum Potsdam, Telegrafenberg, 14473 Potsdam, Germany

Abstract. In the immediate aftermath of a disaster routine processes, even if specifically designed for such a situation, are not enacted blindly. Actions and processes rather adapt their behavior based on observations and available information. Attempts to support these processes by technology rely on process models that faithfully capture process execution and adaptation. Based on experiences from actual disaster response settings, we propose to specify an *adaptive process* as a set of scenarios using a Petri net syntax. Our operational model provides an adaptation operator that synthesizes and adapts the system behavior at run-time based on the given scenarios. An example illustrates our approach.

Key words: process model, adaptation, scenarios, Petri nets, disaster response

1 Introduction

The fairly general notion of a process as a computational entity in a distributed system has found its application, among others, in workflows and process management systems. Here a *process model* relates atomic tasks, data and resources; the runs of the model mimic the behavior observed in reality. A correct model aids in planning and running complex processes while optimizing the use of resources. The validity of the model depends on assumptions, e.g. about the availability of data and resources and the possibility to enact tasks in a specific way.

Disaster response comprises all behavior immediately after a disastrous event with the aim of preventing loss of life and restoring order [5, pp.57]. Even at an intuitive level of understanding it becomes apparent that assumptions of a process or its model are likely to be violated in such situations; deviations are the rule. An actual process in this context is *adaptive* and reacts on critical changes in its environment by changing the process itself as it is being executed.

A feasible model of such a process must not only mimic these adaptation dynamics, called *instance level changes* [24]. The process model itself must be adaptable at run-time if the overall dynamics require sufficiently different process behavior, i.e. *type level changes* [24]. For this reason the model has to be *humanly conceivable* at any point in time to allow these changes.

* The author is funded by the DFG-Graduiertenkolleg 1324 “METRIK”.

In this paper, we present a novel approach for modeling adaptive processes. Rather than changing a static process model we propose to adaptively synthesize process behavior from scenarios. A *scenario* specifies a possible course of (future) actions and the therein involved resources in the context of a larger system. Whether a scenario suits a given situation is specified in a behavioral precondition. A process model then is a set of scenarios describing sequentially connected, concurrent, mutually exclusive, and overlapping behavior.

A run of the model is synthesized during execution by concatenating, merging and removing scenarios depending on observed behavior and their respective preconditions. Moreover, the set of scenarios that participates in the synthesis can easily be changed during the run. Our approach has a formal, operational semantics based on Petri nets [23]. Petri net-based process models are used in existing process management systems, for instance YAWL [3]. Our approach reflects some requirements from the disaster response domain, but our results should extend to other domains as well.

The remainder of this paper is organized as follows. In Sect. 2 we provide an introduction to disaster response and requirements for process models in this setting. In Sect. 3, we present our approach for scenario-based process models and sketch its possible use for process adaptation in Sect. 4. We discuss our model regarding literature and conclude in Sect. 5.

2 Requirements for acting during disaster response

Based on common sense and media, one has an intuitive understanding of the characteristics of a disaster and of its implications of acting in the immediate aftermath of a disastrous event. In the following we want to substantiate this understanding based on results from published literature and our experiences in an ongoing case study.

2.1 A requirement analysis for disaster response

In a requirements analysis based on nine empirical studies of disasters and disaster responses Jul [16] reflects American disaster management practices and classifies working in disaster response. She investigates how the characteristics of a disaster event influence acting in a response. We summarize those of her results which we consider as related to processes and process models.

Jul has found that because the knowledge about an event may arrive only as response is underway (or even later) the number and kind of involved organization changes. A notable effect is that *‘established organizations may partner to form emergent organizations to address specialized demands.’* In these cases suitable operational structures and procedures are developed as the event evolves. [16, pp.140]

Individual responders may have to enact *agent-generated* tasks (based on their skills and experiences) as wells as *response-generated* tasks (due to a specific situation) concurrently. Depending on the event’s *anticipability*, *‘responders*

as well as responding organizations may need to develop new procedures and structures, and may be working in unexpected settings. [16, pp.144]

Jul concludes from her analysis, among others, that response technology (1) *'should seek to aid response-driven tasks, such as planning, coordination and resource management'*; and (2) *'must, insofar as possible, allow flexibility and deviation in their application'* while *'imposing standard structures and procedures'* [16, pp.145]. We could confirm her findings in a concrete case-study.

2.2 Specific requirements from a concrete case-study

The authors conduct a joint case study with the "German Task Force for Earthquakes" co-ordinated by the GeoForschungsZentrum Potsdam, the German research center for geosciences. The task force *'was founded on March 22, 1993, jointly by geoscientists, civil engineers, sociologists, search and rescue specialists, and experts from the insurance industries. The major purpose of the [task force] is to co-ordinate the allocation of an interdisciplinary scientific-technical expert team after catastrophic earthquakes worldwide.'* [21]

The case study aims on (1) understanding actual work procedures of the task force during disaster response (2) finding which aspects can be supported by process models, and (3) propose suitable concepts and technology to do so. The case study is still ongoing, but we want to explain our findings so far confirming and extending the analysis of Sect. 2.1.

The task force is activated immediately after a natural disaster has occurred, collects necessary information about the event and its location, decides about, and prepares for an in-field mission. In case of a mission, at most a week after the initial event, task force members begin collecting and analyzing data at the disaster site including seismic data of aftershocks, post-seismic deformation, hydrogeological data, and the damage distribution as well as structural conditions of buildings. The results support local decisions and provide a scientific basis for an improved intermediate and long term mitigation of earthquake effects, and improve existing theories and their application.

At the site the team members naturally follow their agent-generated tasks for which they are qualified by their expertise as a geoscientist or engineer. At the same time they enact response-generated tasks; this involves for instance organizing accommodation, transport, and communication or maintaining mission critical equipment. By cooperation with other organizations like disaster response groups or military personnel the team members gain professional and organized support. In turn they can use their experienced background to support rapid decisions in coordinating rescue and damage mitigation efforts.

We are currently developing a process model. We succeeded in developing a static model for the initial preparation phase; the current model has about 120 tasks. But we found that this kind of model is infeasible for specifying the work at the disaster site for the following reasons: (a) Tasks and sub-processes are event driven. (b) A task may vary on its prerequisites, effects, duration, and costs. (c) Sub-processes (sometimes executed repeatedly) overlap and synchronize on common data and tasks. (d) As team members work spatially distributed

and communication is unreliable, decision making is also based on the possibility to make them jointly, or not. (e) Agent-generated tasks and response-generated tasks interfere with each other; each team member has to come up with a suitable, individual procedure as the situation evolves.

This result is not surprising in general and confirms a working hypothesis of most works on process adaptation and models for adaptive processes, e.g. [4, 6, 8, 22]. However, we were able to model well-structured fragments of the process: The task force members could provide information about their working through story-telling. Such a story isolates a specific aspect of the process dynamics and is part of the “acting recipes” of the task force members. These recipes are chosen, instantiated, and followed in a concrete context. We turn this fact into a process modeling paradigm in the following and discuss its features wrt. existing approaches thereafter.

3 Oclets – a formal model for adaptive processes

The requirements analysis in the previous section shows that the kind of processes we wish to model is subject and object to very complex dynamics. A static process model is likely to be incomplete and will exhibit a high complexity. Scenario-based specifications are one possibility to diminish behavioral complexity. We propose to adopt this paradigm for process models subsequently.

3.1 Scenario-based specifications

The paradigm of scenarios is widely accepted in protocol specifications using *message-sequence charts*. *Life-sequence charts* (LSCs) take this paradigm one step further by decomposing *behavior* of highly complex, distributed systems into reasonably-sized artifacts. [14]

An LSC specification consists of a set of charts, each specifying a scenario. A “normal” LSC denotes a partial, partially ordered run of a system over local events and exchanged messages. An *anti*-LSC specifies behavior that must not occur while allowing all other. A LSC has a *behavioral precondition* denoting which events have to be observed in which (partial) order to *activate* the chart. An active chart is violated if the system exhibits behavior that is not specified in the chart. A set of LSCs specifies the set of all runs where these charts are not violated.

The LSC technique is declarative in the sense that a specification classifies an observed run as valid or invalid. It has no operational semantics that allows to generate the set of all valid runs of an arbitrary LSC specification; though this is possible for subclasses [13].

3.2 Oclets – adopting scenarios to Petri nets

We want to use the paradigm of scenarios to model highly dynamic processes. We adopt the concepts of LSCs to Petri nets and supplement them with an

operational semantics that allows constructing all valid, partially-ordered runs of the system. In [11] we provide a formal model of our approach together with a proof of correctness. Subsequently, we given an intuitive explanation of the concepts and operations. A basic understanding of place/transition Petri nets will be helpful; an introduction is given in [23].

Oclets. We formalize a scenario as an *oclet* which is a finite, acyclic, labeled Petri net with some further annotations; Fig. 1 depicts some examples. As usual, a box depicts a transition which, in an oclet, denotes the *execution of a task*. A circle depicts a place, here denoting the *presence of a physical or virtual resource*. Arcs denote which resources are consumed and which resources are produced by executing a task. In this interpretation, an oclet denotes a scenario.

An oclet's *precondition*, a causally closed prefix, is depicted above the dashed line. It denotes the behavior that has to be observed to *activate* the oclet. The remaining part of the oclet describes its *contribution*: a *normal oclet* denotes a possible further execution while an *anti-oclet* disallows the execution; see Fig. 6.

In the following we construct complex behavior from oclets by repeatedly concatenating, merging, and removing oclets based on their preconditions. An *adaptive process* $A = \langle O, P_0 \rangle$ is a set O of oclets with a set P_0 of labeled places (describing the initial state). As an example consider the oclets of Fig. 1 and place p_{idle} with label *idle* constituting $A_1 = \langle \{o_1, o_2, o_3\}, \{p_{\text{idle}}\} \rangle$; we denote a place or transition by its label subsequently.

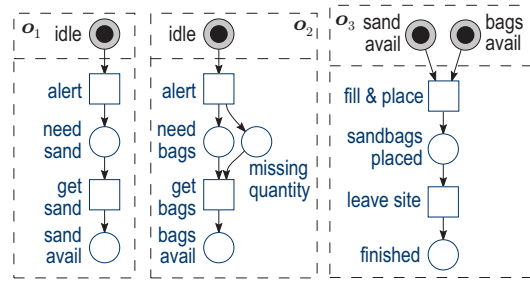


Fig. 1. Oclets o_1, o_2, o_3 of the adaptive process A_1

may leave the site. Note that this example is not based on concrete data and might be too simplistic for an actual process. We extend it as we proceed in explaining our concepts.

3.3 Constructing partially-ordered runs from scenarios

We describe the behavior of an adaptive process by branching processes. Informally, a *branching process* (BP) relates to a partially ordered run like an execution tree relates to a sequential run: A sequential run denotes process behavior as a sequence of tasks by interleaving concurrent tasks; merging sequential runs at common prefixes yields an execution tree. A *partially ordered run* explicitly

A_1 specifies a process of a response team in case of a flooding alert to build a sand bag barrier. A first requirement is to get sand after an alert (o_1). The team shall also get bags according to the missing quantity (o_2). Oclet o_3 denotes that as soon bags and sand are available, the bags shall be filled with sand and placed. After the sandbags are placed, the team

denotes concurrency of tasks. Hence a BP is a directed acyclic graph that distinguishes causal ordering, concurrency and conflict of tasks. It is an important result in Petri net theory that a BP of a Petri net N can itself be represented as a Petri net β_N s.t. any run of β_N is a run of N . [9]

Fig. 2. Initial state s_0 net-BP β together with a reachable marking m of places of β . We explain this notion of state and its relation to oclets in the following. The *initial state* of an adaptive process $A = \langle O, P_0 \rangle$ is the BP consisting of the places P_0 that marks each place $p \in P_0$ with one token, $m(p) = 1$. In our example, the initial state s_0 of A_1 consists of the marked place p_{idle} only, see Fig. 2.

The BP of a state $\langle \beta, m \rangle$ can be extended or reduced by *applying* an oclet o which yields a step $\langle \beta, m \rangle \xrightarrow{o} \langle \beta', m \rangle$. Since the BP is a Petri net, its marking m may enable a transition t of BP allowing the step $\langle \beta, m \rangle \xrightarrow{t} \langle \beta, m' \rangle$.

Adding behavior with state-based preconditions. An oclet o that has a precondition consisting only of a set of places is *enabled* in a state $\langle \beta, m \rangle$ if the precondition of o is contained in the current marking m . To be precise, we need an injective mapping h from equally labeled places of the precondition of o into a set Y of marked places of β . We call Y *enabling set* and h an *embedding* of o 's precondition into β . Our example oclets o_1 and o_2 are enabled in s_0 of Fig. 2

If o is enabled in state $\langle \beta, m \rangle$ then we may perform the *adaptation step* $\langle \beta, m \rangle \xrightarrow{o} \langle \beta', m \rangle$ by adding the contribution of o , i.e. o minus its precondition, to β . Formally, append a copy of o 's contribution to the enabling set and iteratively merge a newly added node with an existing node if both have equal labels and the same predecessors. This may result in merging all new nodes with existing ones. Our formal definitions in [11] are more involved and attribute an oclet as enabled only if the corresponding adaptation step changes β ; here we continue at a more intuitive level.

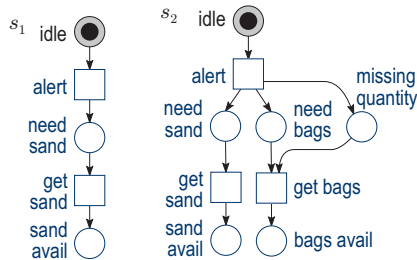


Fig. 3. The step $s_1 \xrightarrow{o_2} s_2$ of A_1 .

In our example, oclets o_1 and o_2 are enabled in s_0 . The result of the step $s_0 \xrightarrow{o_1} s_1$ is depicted on the left in Fig. 3. In s_1 , o_2 is still enabled and the step $s_1 \xrightarrow{o_2} s_2$ adds o_2 and merges the transitions *alert* that originate in o_1 and o_2 , respectively; see Fig. 3. In s_2 transition *alert* is enabled and may fire: The *transition step* $s_2 \xrightarrow{\text{alert}} s_3$ consumes the token on *idle* and produces a token on *need sand*, *need bags*,

and *missing quantity* (which is standard Petri net semantics).

For a consistent semantics, we prioritize adaptation steps over transition steps and require to *apply all enabled oclets* before (nondeterministically) firing one enabled transition. With this definition, we could easily construct and complete the (only) run of our example model A_1 .

Now assume that in our example it may actually happen that the response team can only **get some bags** instead of all, but that it continues with the process afterwards. Oclet o_4 of Fig. 4 specifies this behavior. We include o_4 in our adaptive process; o_4 is enabled in s_3 and the step $s_3 \xrightarrow{o_4} s_4$ adds **get some bags** in conflict to **get bags**, see Fig. 4. Both transitions are enabled in s_4 , but only one of them may fire. The steps $s_4 \xrightarrow{\text{get some bags}} s_5 \xrightarrow{\text{get sand}} s_6$ mark sand avail and bags avail which enables o_3 to finish the process.

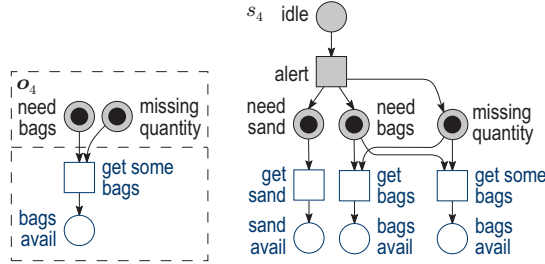


Fig. 4. Oclet o_4 and state s_4 after applying o_4

would rather assess the *history* of its actions and *adapt forthcoming tasks and behavior* if necessary.

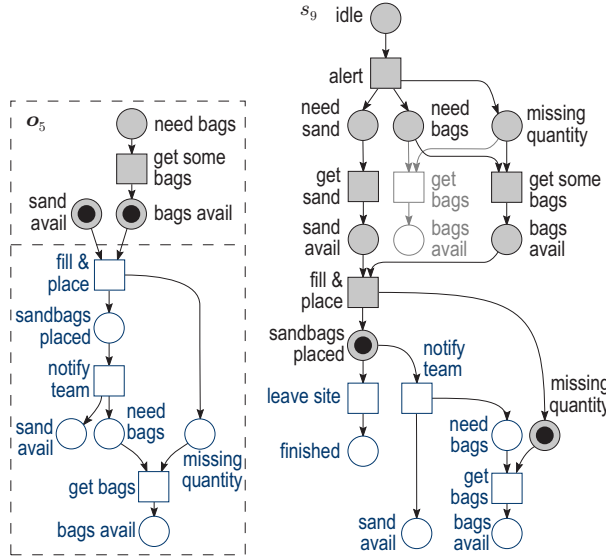
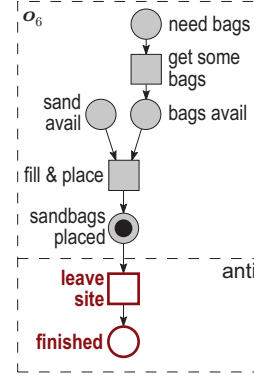
Oclet o_5 of Fig. 5 specifies the following scenario: If only some bags could be retrieved, the response team determines the **missing quantity** as the bags are filled and placed. Then the team is notified that sand is still available but some more bags are needed; the team gets the latter.

Adding behavior with history-based preconditions. The precondition of oclet o_5 in Fig. 5 is more complex and also denotes a partial execution that must have led to the current state in order to activate the oclet. Generalizing our earlier definition, an oclet o is *enabled* in a state $\langle \beta, m \rangle$ if the entire precondition of o is contained in β (including arcs) s.t. the marked places of o are contained in the current marking m . Formally, the *enabling embedding* h of o into β has to be an arc-preserving injection between equally labeled nodes of o and β . The definition of an adaptation step remains as before.

If we include o_5 in our adaptive process, o_3 and o_5 are enabled in state s_6 . If we had fired **get bags** instead of **get some bags** oclet o_5 would not be enabled. The mandatory adaptation steps $s_6 \xrightarrow{o_3} s_7 \xrightarrow{o_5} s_8$ yield the state s_8 that enables transition **fill&place**. By the adaptation step of o_5 , transition **fill&place** gets a new post-place **missing quantity** according to our adaptation semantics; the result of $s_8 \xrightarrow{\text{fill\&place}} s_9$ is depicted in Fig. 5. Thus, depending on the history of the process execution the transition **fill&place** has different effects.

Oclet o_5 introduced task **get bags** again which now also depends on task **notify team**. Observe that **notify team** and **leave site** are in conflict. The latter belongs to the “all went well”-scenario of oclet o_3 while the former was introduced by the “missing resource”-scenario of o_5 . Both scenarios make contradicting statements about how to continue after the sandbags have been placed. There are

However, o_3 does not suit the slightly changed process as its enabling depends on the marking of s_6 alone. By strictly looking at the process definition, the response team would finish its work even if the number of bags was insufficient. Thankfully, this is not the case in reality. An actual response team

Fig. 5. Ocllet o_5 and state s_9 after applying o_5 Fig. 6. Ocllet o_6

several ways how to resolve this contradiction while keeping the process sound; we propose one in the spirit of scenarios.

In an actual process, the response team would know that the “all went well”-scenario was violated; it would not leave if only some bags could be retrieved. We use *anti-oclets* as mentioned in Sect. 3.2 to *explicitly forbid tasks and behavior depending on the execution history*. The anti-oclet o_6 in Fig. 6 specifies that after only some bags were retrieved and filled, the team *must not* leave the site. We include o_6 in our example process.

Removing behavior with history-based preconditions. The enabling condition for anti-oclets is essentially the same as before; ocllet o_6 is enabled in state s_9 of Fig. 5.

If an anti-oclet o is enabled in state $\langle \beta, m \rangle$ then the *adaptation step* $\langle \beta, m \rangle \xrightarrow{o} \langle \beta', m \rangle$ removes the contribution of o from β . Formally, extend the enabling embedding h of o 's precondition in β to map a maximal prefix of o into β . These are the nodes of o 's contribution which are actually present in β ; remove them from β and iteratively remove all nodes of which a predecessor was removed.

In our example, the adaptation step $s_9 \xrightarrow{o_6} s_{10}$ removes transition *leave site* and place *finished* from s_9 . Now *notify team* is the only enabled transition (because *leave site* is no longer present). After the step $s_{10} \xrightarrow{\text{notify team}} s_{11}$ places *sand avail*, *need bags*, and *missing quantity* are marked.

For semantical consistency, we first perform *an adaptation step for each enabled normal ocllet* (adding all possible behavior), then *an adaptation step for each enabled anti-oclet* (removing all infeasible behavior) and then *fire one enabled transition* (nondeterministically choosing one possible, feasible behavior).

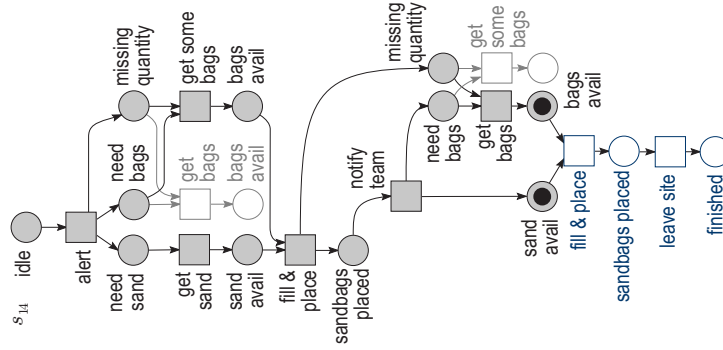


Fig. 7. State s_{14} depicting the result of a behavior synthesis from o_1, \dots, o_6

This definition concludes the basic concepts of our approach for adaptive processes. Let's finish our example: Because o_4 is enabled in s_{11} we have to adapt $s_{11} \xrightarrow{o_4} s_{12}$. Again, the team could either *get some bags* or all bags wrt. the *missing quantity*. The former transition yields a state which is equivalent to s_6 (wrt. execution history): the same oclets and transitions get enabled. *This way oclets express loops in process executions.* We assume that this time the team gets all bags, $s_{12} \xrightarrow{\text{get bags}} s_{13}$ enabling o_3 only, and does not enter the loop again. The result of $s_{13} \xrightarrow{o_3} s_{14}$ is shown in Fig. 7. From there the steps $s_{14} \xrightarrow{\text{fill\&place}} s_{15} \xrightarrow{\text{leave site}} s_{16}$ get the process into its final state, *finished*.

4 Using oclets to realize adaptive processes

We have just presented the basic concepts of our approach for modeling processes with scenarios. In this section we want to sketch how these concepts can be used in modeling adaptive processes.

We already mentioned in the introduction that we understand an actual process to be *adaptive* if it reacts on critical changes in its environment by changing the process itself. This definition is cyclic. Splitting the cycle helps: (1) the behavior of an adaptive process can change at run-time and (2) the change of the behavior is triggered by the process. We explain how our model helps in realizing (1) and spend some ideas on (2).

Considering (1), in our case the *process type* of an adaptive process is given by a set of oclets, see Sect. 3.2. A *process instance* is a state of an adaptive process, being a branching process and a reachable marking as explained in Sect. 3.3. It includes its current run-time state, its execution history, and a part of its future behavior. Our semantic model synthesizes, and extends, an instance from the set of oclets at a time and executes it. Changing the set of oclets changes the future process behavior, as exercised in the previous section.

In our approach, a *type level change* is realized by changing a set – of oclets. Moreover, it smoothly yields *instance level changes* as an instance is incremen-

tally synthesized from a set of oclets at a time. In practice it might be necessary to use instance migration strategies when changing the process type s.t. different instances have different sets of oclets which eventually converge [24].

Regarding (2), our model allows to trigger behavior change in closed systems as follows. Assume two different oclets o_K and o_L having the same precondition where o_K suits some context K and o_L suits some other context L . The adaptive process $A_K = \langle O \uplus \{o_K\}, m \rangle$ shall be feasible in K ; a change to $A_L = \langle O \uplus \{o_L\}, m \rangle$ makes it feasible for L . This kind of change must be triggered externally when someone observes a context change from K to L .

The oclets O can be extended to distinguish contexts K and L ; we demonstrated this in Sect. 3.1 when introducing o_4 . Let O^+ denote these oclets. In the same way, the preconditions of o_K and o_L can be extended differently to distinguish contexts, let this be o_K^+ and o_L^+ ; oclets o_3 and o_5 of Sect. 3.3 are an example. Now the process $A^+ = \langle O^+ \uplus \{o_K^+, o_L^+\}, m \rangle$ distinguishes context K from L and chooses the appropriate behavior. What was an explicit adaptation of behavior from A_K to A_L is now gracefully included in the process dynamics.

5 Conclusion

We have presented an approach to model adaptive processes that can change their behavior at run-time. Such processes arise for instance during disaster response. We presented conceptual and concrete requirements for process models in this setting which exceed the capabilities of static process models.

We proposed to model an adaptive process in scenarios with an intuitively understandable Petri net syntax which we formalized as oclets. We contributed an operational adaptation step that synthesizes the behavior of the process from its oclets at run-time. The synthesis depends on the process execution and may add, remove, or modify future behavior. This effectively leads to a run-time adaptation of process dynamics. Further, the process model can be changed at run-time to adapt to process-external changes. We demonstrated the concepts of our approach in an example from the emergency response settings.

The semantic model for our approach are Petri net branching processes. It is operational and may be verified using efficient model-checking techniques [10].

Discussion and Related Work. The problem of adapting process behavior has been researched from various angles. Many works consider the adaptation of workflows by run-time application of transformation rules on a static process model [4, 6, 8, 22], or by a notion of relating an old system specification to a new system specification to guarantee a correct replacement [1]. Some of these approaches, e.g. [22], are more expressive than our approach. But our oclets make the condition when to perform an adaptation step an explicit part of the operational system specification. This is not the case in the mentioned works, where process adaptation must be triggered externally.

Hee et al. demonstrated how to make process dynamics dependent on execution history in a Petri net model using transition guards [15]. We contribute an

intuitive Petri net-notation for history-dependent dynamics; the actual relation between both models deserves further research.

Petri net theory has developed a number of models that allows operationally changing Petri nets at run-time by putting the net to be changed as a token into a high-level net that does the change [17]. Adaptations have been formalized as direct operations on nets [12] and as graph transformations based on rules that allow constructing system behavior from scenarios [7]. Dynamically adaptive systems can also be expressed with various sorts of process algebras, for instance χ [2] or the π -calculus [20], including communication and changes in topologies. But like in the approaches mentioned above, the operations to adapt process behavior are an explicit part of the process definition. The “worklet” concept of YAWL allows adding encapsulated sub-workflows (“worklets”) during process execution in a hierarchical workflow model [3].

Our result suggests that adaptive process dynamics need no hierarchies and no explicit adaptation operation to be part of the process model. The modeler declares the behavior she wants to include or exclude together with a precondition; the operational behavior is synthesized at run-time. The advantage or disadvantage of having no hierarchies depends on the actual process.

In this direction, Leoni et al [18] propose a situation calculus based on second-order logic to model processes. Their approach includes a control-loop scheme where the process correctly changes its behavior based on actual observations, including unforeseen events. This is not covered by any other approach, including ours. A possible drawback is its text-based specification language that might obfuscate an intuitive understanding of the process.

We already mentioned that the concepts of our approach originate in live-sequence charts [14] which are an entirely declarative technique, while we provide an operational model. We consider a detailed comparison as future work.

We also see another contribution of our approach. The results of Mendling et al [19] suggest that the size of a process model and ‘*the number of arcs [has] an important influence on understandability.*’ They also found that ‘*small variations between models can lead to significant differences in their comprehensibility.*’ We take this as an indicator that our approach of modeling with well-conceivable process fragments has taken a reasonable direction.

Future Work We are currently implementing our concepts and the algorithms in a proof-of-concept run-time environment for adaptive processes. This tool will help us in validating our approach. We are researching the formal properties of our approach that help in verifying such processes. We are also investigating which further concepts we need to include in our model to achieve more expressivity. Benchmarks are our case study from the disaster response domain and practically relevant adaptation patterns as identified in [24].

References

1. W.M.P. v.d. Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, Feb 2002.

2. J.C.M. Baeten and D.A.v. Beek and J.E. Rooda. *CRC Handbook of Dynamic System Modeling*, chapter 19: Process algebra, pages 19.1–21. Chapman & Hall, 2007.
3. M. Adams, A.H.M. t. Hofstede, D. Edmond, and W.M.P. v.d. Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In *OTM Conferences (1)*, pages 291–308, 2006.
4. A. Agostini and G. De Michelis. Improving Flexibility of Workflow Management Systems. In *LNCS*, volume 1806, pages 218–234. Springer-Verlag, 2000.
5. W. Nick Carter. *Disaster Management: A Disaster Manager's Handbook*. Asian Development Bank, 1991.
6. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 438–455, 1996.
7. H. Ehrig, K. Hoffmann, and J. Padberg. Transformations of Petri Nets. *Electr. Notes Theor. Comput. Sci.*, 148(1):151–172, 2006.
8. C. Ellis, K. Keddera, and G. Rozenberg. Dynamic change within workflow systems. In *COCS'95*, pages 10–21. ACM Press, 1995.
9. J. Engelfriet. Branching processes of Petri nets. *Acta Inf.*, 28(6):575–591, 1991.
10. J. Esparza and K. Heljanko. *Unfoldings - A Partial-Order Approach to Model Checking*. Springer-Verlag, 2008.
11. D. Fahland. Oclets - a formal approach to adaptive systems using scenario-based concepts. Informatik-Berichte 223, Humboldt-Universität zu Berlin, 2008.
12. B. Farwer. Recovery and reset in object petri nets with process markings. In *Proceedings of CS&P 2006*, pages 47–57, Sept. 2005.
13. D. Harel and H. Kugler. Synthesizing State-Based Object Systems from LSC Specifications. In *LNCS*, volume 2088, pages 1–33. Springer-Verlag, 2001.
14. D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., 2003.
15. K.M.v. Hee, A. Serebrenik, N. Sidorova, M. Voorhoeve, and J.M.E.M.v.d. Werf. Modelling with history-dependent petri nets. In *LNCS*, volume 4714, pages 320–327, 2007.
16. S. Jul. Who's Really on First? A Domain-Level User, Task and Context Analysis for Response Technology. In *ISCRAM 2007*, pages 139–148, 2007.
17. M. Köhler and H. Rölke. Reference and value semantics are equivalent for ordinary object Petri nets. In *LNCS*, volume 3536, pages 309–328. Springer-Verlag, June 2005.
18. M. de Leoni, M. Mecella, and G. De Giacomo. Highly dynamic adaptation in process management systems through execution monitoring. In *LNCS*, volume 4714, pages 182–197. Springer-Verlag, 2007.
19. J. Mendling, H.A. Reijers, and J. Cardoso. What makes process models understandable? In *LNCS*, volume 4714, pages 48–63, 2007.
20. R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999.
21. GFZ Potsdam. Mission statement of the “German Task Force Earthquakes”. http://www.gfz-potsdam.de/pb2/pb21/Task_Force/index_e.html, 15th May 2008.
22. M. Reichert and P. Dadam. ADEPT_{flex}-Supporting Dynamic Changes of Workflows Without Losing Control. *J. Intell. Inf. Syst.*, 10(2):93–129, 1998.
23. W. Reisig. *A Primer in Petri Net Design*. Springer Compass International, 1992.
24. B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. In *LNCS*, volume 4495, pages 574–588, 2007.

Workflows in Dynamic Development Processes

Thomas Heer, Christoph Briem, and René Würzberger

RWTH Aachen University, Department for Computer Science 3,
Ahornstr. 55, Aachen 52074, Germany,
<http://se.rwth-aachen.de>,
{heer|chrbriem|woerzberger}@i3.informatik.rwth-aachen.de

Abstract. Process management systems are used in many domains to monitor and control processes. Development processes require specific support, which is not provided by conventional workflow systems. On the one hand, they comprise highly dynamic process parts, which cannot be defined until project runtime. On the other hand, development processes may contain subprocesses, which can be executed in the form of predefined workflows. In this paper we describe the integration of a specialized system for the management of dynamic development processes with a conventional workflow management system. This integrated solution targets the specific needs for the management of engineering design processes in the plant construction domain. It has been implemented as an extension to a commercial CAE-tool, and it will be evaluated in industrial practice in the near future.

Key words: dynamic workflow, development process, task net, process management

1 Introduction

Process management is nowadays used in many *different application domains*, which often comprise substantially different *process types*. The managed processes range from claim handling cases in insurance companies to development processes in software or mechanical engineering. The characteristics of a process type lead to *specific requirements* for the respective *process meta-model* and the *tools* for managing processes of this type.

An insurance workflow for example usually describes a *predefined procedure*, which a clerk has to follow. In contrast to that, a development process definition often merely defines the main phases or coarse-grained steps to be executed during the development of a product, as well as the main artifacts, which should be produced. In the former case, language constructs are needed for modeling *alternative courses of action*. Deviations from the predefined process definition constitute exceptional cases. In the latter case the execution of *ad-hoc processes* should be supported. These processes can be *elaborated during enactment*, whereas the process model is continually modified.

While workflow technology can be applied to support the former process type, it is not suitable for the support of development processes as a whole.

Therefore, we implemented the process management system PROCEED¹, which provides *specific support* for the management of *development processes* in the plant engineering domain.

On the one hand, conventional workflows are not appropriate to model whole development processes, because many dynamic subprocesses cannot be predefined before enactment. On the other hand, there are nonetheless many *subprocesses* in a dynamic development process, which are *repetitive and static*. These static subprocesses can be supported by common workflow management systems. But they must not be executed in isolation, unaware of their *surrounding process context*. Therefore we integrated PROCEED with a conventional workflow management system to support dynamic development processes, which comprise predefined workflows as subprocesses.

The development of this integrated solution is undertaken in the context of the *transfer project T6* [1] of the *transfer center 61* of the *collaborative research center 476 IMPROVE* [2]. In this project we transfer the research results of the AHEAD project [3] to industrial practice. The project is funded by the DFG² and is executed in close cooperation with the innotec corporation [4]. PROCEED is implemented as an extension to the computer aided engineering (CAE) tool *Comos*, a product of innotec.

The paper is structured as follows. In Section 2 we describe DYNAMITE meta-model for dynamic development processes and compare it to conventional workflow meta-models. In Section 3 we describe, how we integrated the two different approaches into a system, that supports all types of static and dynamic subprocesses in a development project. In Section 4 we describe and evaluate related work before we give a conclusion in Section 5.

2 Dynamic Task Nets and Conventional Workflows

In this section we briefly describe the DYNAMITE meta-model [5, 6, 3] for the modeling and enactment of *DYNAMIC Task nEts* and compare it to conventional workflow meta-models.

2.1 DYNAMITE

The development of DYNAMITE was motivated by the need for *tool support* for the *management of development processes*. Therefore, the DYNAMITE meta-model comprises language constructs for modeling the coordination of engineers in simultaneous engineering scenarios, the propagation of development artifacts and the like.

DYNAMITE is part of an *integrated meta-model* for the management of dynamic development processes, which constitutes the foundation of the process management system AHEAD [3] as well as of the newly developed PROCEED

¹ PROCess management Environment for Engineering Design processes

² Deutsche Forschungsgemeinschaft

system. While AHEAD was a research prototype, PROCEED is an extension to the commercial CAE-tool Comos. DYNAMITE is the partial meta-model for task management. It is complemented by ResMod (Resource Modeling) and CoMa (Configuration Management), which are meta-models for *resource management* and *product management* (documents, artifacts), respectively. The three partial meta-models are tightly integrated. In the following we briefly describe the modeling constructs of DYNAMITE by example.

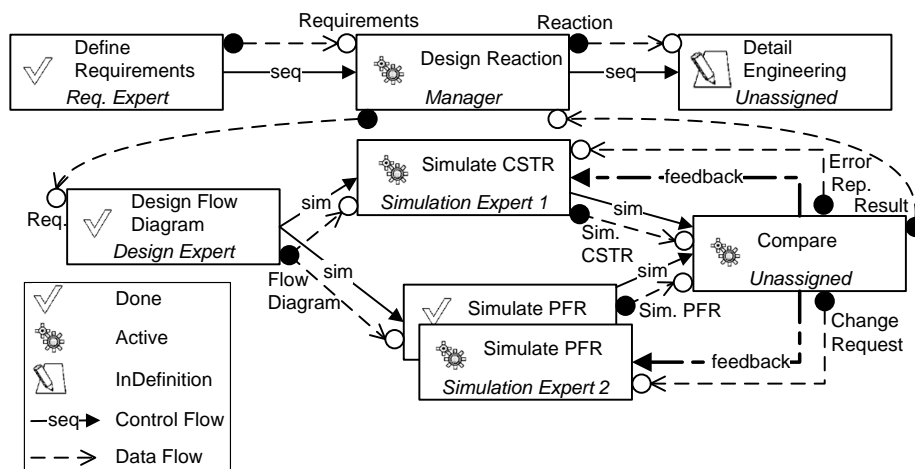


Fig. 1. Example of a DYNAMITE task net.

In Figure 1 a small example of a DYNAMITE task net is depicted. This example has been derived from a larger scenario from the domain of chemical engineering, which is described in [2]. Tasks are represented by rectangular boxes which are connected by *control flow*, *data flow* or *feedback flow* edges.

Six different *execution states* of tasks are defined in DYNAMITE: *InDefinition*, *Waiting*, *Active*, *Suspended*, *Done*, and *Failed*. The transition rules for the execution states of a task are defined in the form of a finite state machine (cf. [5]). The possible execution state transitions of a task depend also on the execution states of all tasks, which are connected to the task by control flows and feedback flows.

In DYNAMITE a *control flow* connects exactly two tasks and it can have one out of three different semantics.

- A *standard* control flow defines, that the target task of the control flow must not be terminated before the source. This is the minimal requirement for a control flow. However, the source and target of a control flow can be executed in parallel.
- A *simultaneous* control flow defines the additional restriction, that the source task must be started before the target. The motivation for this control flow semantic is, that the resources of the source and target tasks cooperate, and

that a first intermediate result of the source task is required to start working on the target task. At the same time the standard semantics guarantees that the last version of the source’s output is consumed by the target task before it is terminated.

- The most restrictive control flow is that of type *sequential*. In this case the target task may only become active after the source task has been terminated. Note, that most workflow management systems only use the sequential control flow semantics and therefore restrict the execution order of tasks too much for typical cooperation scenarios in development processes.

The DYNAMITE meta-model also comprises the concept of *data flows*. Data flows refine control flows. By means of data flows it can be explicitly modeled, which products are transferred from one task to another.

The tasks of a process, which is modeled using DYNAMITE, are all part of one overall task net. This task net is *hierarchically structured*. Each task can contain arbitrarily many subtasks. In Figure 1 the hierarchy is depicted by means of the layout. The subtasks of **Design Reaction** are arranged below the task itself. In PROCEED it is possible to navigate through the task hierarchy.

In DYNAMITE *feedback flows* address the specific dynamic situations in development processes. Although the model of a development process usually defines dependent tasks and hence an *order of execution*, the actual enactment of these tasks in a project is *carried out iteratively*. In case of a change request or a detected error in an artifact, certain *previous process steps* have to be *redone*. To trace cases, in which feedback from a succeeding task is given to a previous task, a feedback flow is introduced. If a task has already been terminated and therefore has to be restarted to handle feedback from succeeding tasks, a *new task version* is created for this and succeeding terminated tasks (cf. Fig. 1). The use of feedback flows and versioned tasks is a possibility to redo already terminated process parts during enactment and to ensure *traceability* at the same time. The former a functionality, which most workflow management systems do not support to this day, but which is needed in dynamic development processes because of changing requirements or late detected errors.

2.2 Comparison with Workflow Meta-Models

In this section we outline the major differences between DYNAMITE and conventional workflow meta-models.

The DYNAMITE meta-model has been developed based on considerably *different requirements* compared to conventional workflow meta-models. It was specifically developed to support the enactment of dynamic development processes. *Concurrent engineering* and *cooperation* of engineers is supported by specific modeling constructs like simultaneous control flows and data flows. The support goes beyond the mere parallel execution of tasks. The *interleaved modeling and enactment* of development processes is supported. *Redo* of already terminated process parts and *feedback* to earlier tasks in the process are possible. *Traceability* of all changes to the management data is provided. The task

management data is *tightly integrated with the engineering artifacts* as well as the resource data of the process.

Altogether, the DYNAMITE meta-model *combines* elements of *project plans* and *workflow definitions*. On the one hand, a work breakdown structure, scheduling data and task dependencies like end-to-end relationships can be defined. On the other hand, the tasks are executable and information flow between tasks can be modeled.

DYNAMITE's capabilities for *modeling executable process definitions* and for automatically enacting these processes are limited. This has been an intentional design decision. Modeling constructs like *alternative* or *loop* have been deliberately left out of the meta-model for the following reasons: First, development processes are *highly creative* processes, which cannot be completely predefined in advance. Second, the resources involved in a development process are nearly *exclusively human resources*, which renders automation of tasks useless. Third, DYNAMITE was developed to support dynamic development processes on a *medium-grained level*. On this level one manually decides for a course of action and fixes the process steps to achieve a certain subgoal. Loops, where in each iteration a different course of action may be chosen, are not common. These kinds of scenarios are typical for *fine-grained personal processes* of individual engineers.

Here lie the strengths of conventional WfMS. Common workflow meta-models contain constructs for alternative branching and loops. But conventional workflow management systems are not suitable for the management of whole development projects. Usually, there are only the Sequence and the Parallel control flow types. The former is too restrictive to handle the simultaneous execution of activities, and the latter is too loose to handle cooperation scenarios. Furthermore, most conventional workflow meta-models do not provide constructs for the modeling of hierarchically structured development processes. For each subprocess in the project there would have to be a workflow definition before the subprocess could be enacted by a WfMS. This is often not feasible in a development project. Redo of process parts can only be simulated by dynamically adding copies of already terminated activities to a running workflow instance. In general, it might be possible to simulate all the needed functionality by using current dynamic workflow management systems, but this would not constitute suitable support for dynamic development processes.

Workflows and processes in general can be arranged on a *spectrum* according to their *degree of flexibility*. This has been done e.g. in [7]. On one end of the spectrum there are *completely predefined static workflows*, which cannot be modified at runtime, and on the other end there are *ad-hoc workflows* without any definition. In the latter case tasks are planned and ordered as needed during the enactment of the process, which is a common case in development projects. The common approach to add flexibility to static workflow management systems is to allow *deviations of running instances* from the workflow definition in the form of structural changes. The resulting flexible workflows have to be distinguished from ad-hoc workflows. The latter can be simulated by the former by

using nearly empty workflow definitions and elaborating them during runtime. But this shows the different concepts behind conventional WfMS and the DYNAMITE approach. The *dynamic changes* are considered as *exceptional cases*, because usually the workflow definition should already allow all common courses of action. In DYNAMITE, dynamic changes to the process model are considered as the *normal case*. The process evolves during enactment³. In that sense a dynamic process in our sense should be understood as an ad-hoc workflow in contrast to a flexible workflow.

Between completely predefined workflows and ad-hoc processes there are all *combinations* of static, flexible and ad-hoc workflows. These cases can be modeled by hierarchically structured processes, containing subprocesses of different types.

While DYNAMITE has proven to be appropriate for the management of most medium-grained subprocesses in a development project, workflow support is needed for recurring personal processes of individual engineers and for structured procedures like e.g. in a change management scenario. We try to overcome this minor deficiency of the DYNAMITE model by integrating PROCEED with our WfMS.

3 Integrating Static and Dynamic Process Parts

In our research project we built a full-fledged WfMS as an extension of the CAE-tool Comos by means of the Microsoft Windows Workflow Foundation (WF) [8]. Our WfMS implements all interfaces of the *workflow reference model* [9] of the *Workflow Management Coalition* and even allows for dynamic changes of workflow instances at runtime. Therefore, it is superior to many commercial WfMS and comparable to most of the research prototypes in the adaptive and evolutionary workflow fields (cf. Sect. 4). In this section we describe the integration of PROCEED with the WfMS based on the Windows Workflow Foundation.

The overall development process is *structured hierarchically* in accordance with the DYNAMITE meta-model. It forms a tree hierarchy of subprocesses. Each subprocess on each level of the hierarchy can either be *enacted manually* or *automatically* by means of a workflow. Figure 2 shows an example of a hierarchical process in an abstract notation. Workflow-managed subprocesses are depicted with rounded edges.

The example is taken from the domain of plant engineering, in which the CAE-tool Comos is used. The design process of a plant is divided into several phases like e.g. *Basic Engineering* and *Detail Engineering*. Several flowsheets have to be created like a process flow diagram (PFD) and a piping and instrumentation diagram (P&ID). Devices, which are placed on the flowsheets, have to be specified, which can be done according to a predefined procedure defined

³ Here, process evolution is understood with a different meaning compared to continuous improvement of a process definition.

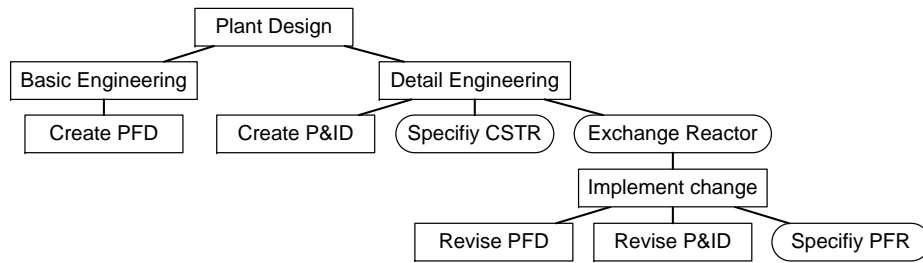


Fig. 2. Cutout of an example plant design process.

in the form of a workflow. There are different types of reactor devices like e.g. a continuously stirred tank reactor (CSTR) or a plug flow reactor (PFR).

While the flowsheets of the plant design are elaborated, it may occur that a task like **Exchange Reactor** in figure 2 is created. It defines that a certain CSTR should be replaced by a PFR in the plant design. For this purpose, a predefined *change management workflow* is started, which realizes the task **Exchange Reactor**. Whereas the *general procedure* for this change request is fixed, the *details are specific* for exchanging the CSTR in question, and they cannot be defined in general.

A simplified version of the change management workflow is depicted on the left hand side of Figure 3. It contains four activities⁴, which define the necessary steps to handle a change request. The activity **Implement change** of this workflow instance contains tasks which are specific for the exchange of CSTR 47 by PFR 11. Therefore, the realization of the activity is a dynamic task net which is defined during the execution of the activity **Plan change**. The subtasks of **Implement change** together with the defined control flows form a typical engineering process. All tasks can be executed in parallel. Only the completion events of the tasks are restricted by simultaneous control flows, e.g. the revision of the P&ID must not be finished before the revision of the PFD, because the former depends on the latter. DYNAMITE provides the necessary means to model the subprocess **Implement change**, which would not be possible by means of the workflow management system. The strengths of the WfMS come into play, when there is a need for control structures like loops, here indicated by the backward arrow from **Review change** to **Implement change**. As long as the changes are not approved, some of the tasks of **Implement change** have to be re-iterated.

The process management component and the workflow management system are *tightly integrated* when it comes to the execution of workflows within a development process, which is illustrated by the following examples. When a DYNAMITE task, which should be controlled by a workflow, like **Exchange Reactor** becomes active, then a workflow instance is created and started via the WfMS. After this workflow instance has been successfully terminated, the task state is changed to Done. Within the workflow instance the control flow must reach **Im-**

⁴ In the following, an element of a dynamic task net is called a *task* and the according element in a workflow definition is referred to as an *activity*.

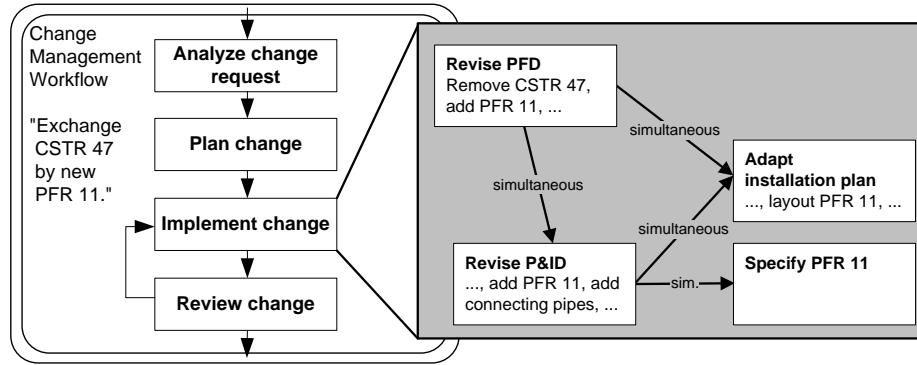


Fig. 3. Change management workflow for task Exchange Reactor.

plement change before the subtasks of this activity can be started. Furthermore, the workflow cannot proceed with Review change before the whole subprocess of Implement change has been successfully terminated.

Workflow-managed Dynamic Task Nets. A human process manager can freely modify a task net, where only certain general consistency constraints apply. However, when a task is controlled by a workflow, the WfMS serves as an *automatic manager* for this subprocess and replaces the human manager to some extent. The subtasks of a *workflow-managed task* are contained in a dynamic task net. But the creation and deletion of subtasks and control flows as well as certain state changes of subtasks are exclusively done by the WfMS, and it is therefore not possible to manually edit the dynamic task net.

The task net of a workflow-managed task contains a *corresponding subtask* for every activity in the workflow definition. These subtasks are connected by sequential control flows according to the workflow definition and the current execution state of the workflow. In Fig. 3, the corresponding task net of the workflow is not shown.

The dynamic task net always reflects the *current state* of the running workflow instance. It comprises the *followed execution path* of the workflow instance as well as *future tasks*, which may be scheduled for execution.

Loops in the workflow are *unfolded* in the dynamic task net. The iterations of a loop are reflected by a corresponding sequence of tasks or – if the loop contains complex control structures – by a sequence of complex subnets.

Until the condition of an *alternative branching* is evaluated, the tasks corresponding to the activities of a branch are connected by control flows with each other, but the resulting subnet is not connected to preceding tasks. When one of the branches is entered, then the control flows to the preceding tasks are automatically inserted. The neglected branches of the alternative branching are only removed after the termination of the workflow.

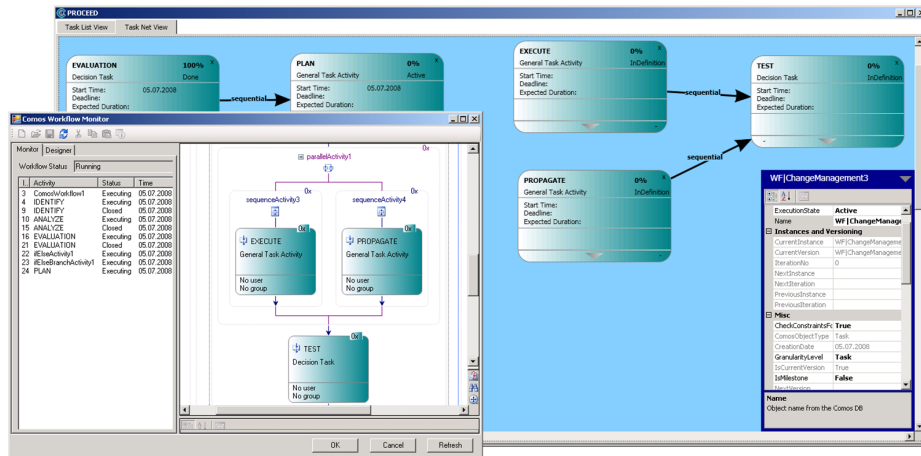


Fig. 4. Screenshots of PROCEED and the workflow monitor.

We do not go into details here, because the similar problem of creating project plans according to the execution of workflows has been investigated in related work already (cf. Sect. 4).

Manual changes to a workflow-managed task net can only be achieved by *changing the definition of the running workflow instance*. Our WfMS allows these kind of dynamic changes. When the workflow definition is changed during the execution of a workflow, then the *corresponding task net is automatically adapted*. If an activity has been added or removed from the workflow definition, its corresponding task is added or removed from the dynamic task net, respectively. The control flow relations between the tasks are adapted accordingly.

Figure 4 shows a screenshot of PROCEED and the workflow monitor of the WfMS. The task net view of PROCEED shows the dynamic task net, which is controlled by the monitored workflow instance. PROCEED currently comprises a work list view and a task net view. Views for resource management and project status analysis are currently under development. PROCEED has been integrated with MS Project. We use e.g. the Gantt-chart view of MS Project to present the coarse-grained tasks of the development project to the user. Changes made to the management data in either PROCEED or MS Project are directly reflected in the other system, respectively.

4 Related Work

There are two major research areas that are related to the contents of this paper. First, there are plenty of research works dealing with *flexibility* in processes that are supported by some workflow management system. Second, also the *integration of project and workflow management systems* is an issue that has been dealt with by diverse research groups.

Flexibility. In many domains there are processes which cannot be modeled completely before execution. Thus, lots of works have been devoted to the issue of process flexibility. Concerning this matter, Weber et. al. provide an overview in [10]. We consider the issues of schema evolution, version control and instance migration as relevant for standardized workflows in engineering design processes. However, the most important issues in this context are support for the enactment of ad-hoc processes and the traceability of changes. These issues were addressed by the development of dynamic task nets.

In [7] a classification of workflows regarding their degree of flexibility. The different workflow types range from ad-hoc workflow without any definition to completely predefined static workflows. The paper concentrates on business processes and office workflows supported by CSCW technology. We partly agree with the authors that ad-hoc workflows have been viewed as not worthwhile to be automated in the past. This is probably true for rather short-lived workflows. But the support of ad-hoc subprocesses as part of complex development processes has been studied for years [3].

Casati et al. advocate the automated handling of unforeseen process *exceptions* using an “exception-specification language” [11]. This language provides additional declarative set-oriented conditions which complement common imperative workflow definitions. Apparently, their approach targets at processes for which automated exception handling is feasible. Since development processes do not call for complete automation and are very creative, we think that changes are better done manually for these kinds of processes.

Within the ADEPT project Reichert et al. investigate dynamic changes to running workflow instances, which are represented by graphical “control flow graphs” [12]. At this, they use a graph based calculus with some optimizations to ensure the correctness of control flow graphs. Though support for dynamic changes is a crucial requirement for development processes, the ADEPT approach does not entirely match our needs. For example, ADEPT control flows are not suitable for modeling dependencies between task that are supposed to be executed in a “simultaneous engineering” mode. Furthermore, the ADEPT meta model rather focuses on the activity aspect of processes but does not yet deal with complex product and resource structures, which are also relevant in development processes.

Integration of project and workflow management. The approach described in this paper is closely related to the problem of integrating a WfMS with a project management system (PMS). In [13] the IPPM system is described which integrates features for both project and process management. An approach for the unfolding of workflow control structures to tasks in a project plan is presented. We followed a similar approach, but we implemented a slightly different branching method, where neglected branches are not removed from the task net before workflow termination, and we do not need to insert estimation tasks for expected loop iterations because our workload and progress estimation for a workflow-managed task works independent of its subtasks in the task net.

In [14] Bussler discusses issues regarding the integration of WfMS and PMS in general. He distinguishes two parts: schema integration and behavior integration. Regarding schema integration, the main conflicts and different possible mappings are discussed. We encountered similar problems, since the elements of the DYNAMITE meta-model are closely related to the common concepts in project management, e.g. there is no conditional branching but an end-to-end task relationship. We decided to restrict the usage of modeling constructs in workflow-managed task nets. For the mapping of control structures we used the continuous mapping approach for loops and the static mapping approach for alternative branching. Regarding behavior integration, Bussler describes an ideal integration independent of any specific system. In our case the major problems arose from the peculiarities of the specific systems at hand.

Bauer addresses in [15] the issue of integrating existing workflow and projects management systems. He distinguishes two approaches: loose coupling, where several workflow instances can be mapped to a single project task, and close coupling, where there is a one-to-one mapping of workflow activities and tasks in the project plan. The close coupling approach is not applicable, when the project plan and the workflows are on different abstraction levels. Hence Bauer presents a generic integration architecture for loose coupling based on an integration layer between the WfMS and the PMS. The propagation and aggregation of runtime data via the integration layer is specified by means of event-condition-action(ECA)-rules. In our research project we realized a loose coupling between PROCEED and MS Project and a close coupling between PROCEED and our WfMS. PROCEED serves as an integration layer between the WfMS and the PMS, but instead of using ECA-rules the connection between the project plan and the workflow instances is defined by a dynamic task net, which represents the whole development process.

5 Conclusion

In this paper we proposed DYNAMITE as an appropriate process meta-model for dynamic development processes. We compared it to conventional workflow meta-models and identified the respective strengths and weaknesses. To support the whole spectrum of dynamics in development processes, we integrated the PROCEED system with our conventional WfMS. A hierarchically structured process with interleaved static predefined and dynamic ad-hoc subprocesses can be enacted. The integrated system has been implemented as an extension of the commercial CAE-tool Comos of the innotec corporation. Altogether, we advanced research results of the CRC 476 IMPROVE [2] and transferred them to industrial practice. The new process management functionality of Comos will be evaluated by selected customers of innotec.

References

1. Heller, M., Nagl, M., Wörzberger, R., Heer, T.: Dynamic Process Management Based Upon Existing Systems. [2] 733–748
2. Nagl, M., Marquardt, W., eds.: Collaborative and Distributed Chemical Engineering: From Understanding to Substantial Design Process Support. Volume 4970 of LNCS. Springer, Berlin (2008)
3. Heller, M., Jäger, D., Krapp, C.A., Nagl, M., Schleicher, A., Westfechtel, B., Wörzberger, R.: An Adaptive and Reactive Management System for Project Coordination. [2] 307–373
4. innotec corporation: Website (2008) <http://www.innotec.de/en>.
5. Krapp, C.A.: An Adaptable Environment for the Management of Development Processes. PhD thesis, RWTH Aachen University, Aachen (1998)
6. Nagl, M., Westfechtel, B., Schneider, R.: Tool Support for the Management of Design Processes in Chemical Engineering. Computers and Chemical Engineering **27**(2) (2003) 175–197
7. Huth, C., Erdmann, I., Nastansky, L.: GroupProcess: Using Process Knowledge from the Participative Design and Practical Operation of Ad Hoc Processes for the Design of Structured Workflows. In: Proc. of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34), IEEE Conference Publishing Services (2001)
8. Microsoft: Windows Workflow Foundation (2008) <http://msdn.microsoft.com/en-us/netframework/aa663328.aspx>.
9. Workflow Management Coalition: The Workflow Reference Model, <http://www.wfmc.org/>. (January 1995) Document Number WFMC-TC00-1003, Issue 1.1.
10. Weber, B., Rinderle, S.B., Reichert, M.U.: Change Patterns and Change Support Features in Process-Aware Information Systems. In Krogstie, J., Opdahl, A.L., Sindre, G., eds.: Proceedings 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007), Trondheim, Norway. Volume 4495 of Lecture Notes in Computer Science (LNCS), London, Springer (June 2007) 574–588
11. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and Implementation of Exceptions in Workflow Management Systems. ACM Transactions on Database Systems **24**(3) (1999) 405–451
12. Reichert, M., Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control. Journal of Intelligent Information Systems **10**(2) (1998) 93–129
13. Chan, K., Chung, L.: Integrating Process and Project Management for Multi-Site Software Development. Annals of Software Engineering **14** (2002) 115–143
14. Bussler, C.: Workflow Instance Scheduling with Project Management Tools. In: DEXA '98: Proceedings of the 9th International Workshop on Database and Expert Systems Applications, Washington, DC, USA, IEEE Computer Society (1998) 753
15. Bauer, T.: Kooperation von Projekt- und Workflow-Management-Systemen. Informatik - Forschung und Entwicklung **19** (2004) 74–86

Practical Experiences of Emergency Management

Dirk Hagebölling

Civil Defence Authority
City of Bochum
Germany
hageboelling@bochum.de

Abstract. This short paper summarizes the invited talk given at the workshop. The main topic has focused on personal experiences with using Information Systems during emergency management. Specifically, the talk has covered the current situation of the systems for emergency management as well as what is expected additionally from the “next generation” system for emergency management to improve effectiveness and efficiency in the response.

Keywords: Emergency Management, Medical Assistance, Computer-aided tools.

1 Introduction

Emergency management is always associated with time critical processes and resource restricted operations. In this context the short run availability of relevant information is very important for an effective handling. Nearly every emergency starts with a telephone call or the reception of an automatic signal by a control centre.

After evaluating the message, basic information must be available about type, extent and location of the specific situation. These are the minimum premises to open an emergency operation by a dispatcher and it is quite equal if he has to deal with a fire, rescue demand or medical distress. The alarm procedure to respond the emergency follows in general a prepared template. In most of all cases additional information after this can only received by the control center after inquiries at the scene. Only some information is permanent available in form of stored or external datasets. These are mostly geographical or geo-related information. An evaluation of dynamic characteristics belonging to the scene will be taken into consideration perhaps by looking at the weather forecast. After finishing the operation, statistic data have been registered, often manually sometimes by using reading devices for barcodes or magnetic stored information.

2 Relevant fields for the application of Computer-aided tools.

Relevant fields of Computer-aided tool applications can be explained by a short description of typical scenarios.

2.1 Medical Emergencies

A team of Emergency medical services (EMS) is running out to a reported medical distress. Only some information about the condition of the client can be won by the call. The emergency case will be classified by dispatcher as routine transport or urgent therapy. After searching for a unit near by the scene (e. g. with support by GPS) relevant information will be sent back by radio systems. At the scene the sick person will be inspected by the EMS' team, which has to decide about the first treatment and to choose an appropriate hospital for taking over

During this, a well structured medical care system is waiting passive in the background because necessary information could not be transferred multilateral. On the other hand the EMS teams have no chance to get an overview, where in hospitals vacant capacities are available, which can be reached in time.

Information is needed on site and at back end. Such information is partly transmitted by telemetric systems (EKG, blood pressure, oxygen saturation etc.) from the operation team at the scene to a hospital or rather by one-line access arranged in the ambulance vehicle to the database of relevant hospitals, control canters or heliports.

Only by practicing a dynamic transfer of person- and system-related information among different partners and institutions in order to prepare the following steps of transport and treatment the involved persons are able to work out the best method of handling the situation. This could be ensured by so-called "MANETs" (mobile ad hoc network). Nowadays these procedures are still restricted by missing (valuable) options for telemetry and protected databases, organized in specific PMSs. A MANET is a peer-to-peer network of mobile devices (such as PDAs and laptops) that are connected with each other through Wi-Fi-based links. Nodes in radio-range can communicate directly, whereas non-neighbouring nodes can use intermediate devices as relays.

2.2 Fire alarms

In case of a fire the alarm the following procedures are comparable to other emergency responses. There is also a background system waiting to support the acting teams.

But the job of those teams can be more difficult, because different static and dynamic information is required at the scene.

- Information (technical) about the burning object, accesses, specific hazards
- Information about persons in danger

- Information about water supply
- Information about own resources and consumptions at the fire ground
- Information about the status of operation teams during structural fire fighting (temperature, air-flow of breathing apparatus, heart rates etc.

It would be helpful in every case, whether information about in-danger people could be provided by technical system (sensors), which offer information at least about the present health state (e.g. carboxyhemoglobin). The extent of the fire could be estimated, when temperature and smoke sensors are monitoring continuously the different floors in the burning house.

But these are almost data, which are needed at the scene. In the background logistic aspects are more important. A permanent control of consumption could insure a replacement in time.

Using data processing by system management for example with methods of critical path planning (CPM) lacks at the scene can be excluded.

It can not be closed out, that fire fighters get lost whilst to struggling inside of buildings.

The combination of Personal Protection Equipment (PPE) with RFID-chips could be useful for necessary search and rescue activities.

Moreover sensors connected to the different parts of PPE can record different exposures in order to decide later on about replacements.

2.3 Emission of harmful chemicals

During such emergencies the monitoring of the scene in relation to changing weather conditions is necessary to estimate trends and developments. Not only operation-teams at the scene need such an information, but also disaster management staffs in the background is, however, interested to work out strategic planning for warning an evacuating endangered parts of local population.

In case of clearing larger areas, evacuated persons have to be registered and the movements should be documented in databases with an enabled access for relevant institutions and authorities.

One method for handling the upcoming challenges can be the attachment of RFID to removed persons and a strict data recording at every station passed by.

3. Final Remarks

Emergency management has a high importance in modern societies. In the light of the aforementioned three typical examples of nowadays' emergency management leads to the conclusion that there is a real need for assistance by Computer-aided tools in general and PMSs in particular.

Currently civil operators use computer aided-systems to activate, control and guide emergency teams. On the other hand they are depending on specific information about buildings, objects, transmission ways, infrastructures etc. in urban areas.

Some parts of this information are stored in a central data system for PMS, some other information is provided separately for "running-out" teams by portable tablet-PCs, supporting the team leaders.

We have recognized that there occurs a mainly demand on geo-related information for strategic planning and in future the evaluation of sensor data will be more and more relevant to guide emergency operations more dynamic than today. This opens an interesting field for future research directions.

Domain-driven Process Adaptation in Emergency Scenarios

Marcello La Rosa¹ and Jan Mendling²

¹ Queensland University of Technology
126 Margaret Street, Brisbane QLD 4000, Australia

² Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
m.larosa@qut.edu.au, contact@mendling.com

Abstract. Business process models are particularly suited to efficiently handle the coordination of tasks in emergency scenarios where goal achievement is very time-critical. When a new emergency becomes imminent, a generic business process for disaster management is to be adapted to the specific requirements of the new scenario. However, not only this adaptation process requires a thorough understanding of the domain in question, but also of the modeling language which the process has been constructed in. This may hinder the involvement of domain experts, who are usually not proficient in modeling notations, but, on the other hand, should ultimately benefit from an efficient coordination of the actions to be taken. To facilitate the adaptation of process models, we propose a questionnaire-driven approach based on a language-independent representation of domain choices and their dependencies. The approach relies on the assumption that every new scenario can be determined by the combination of a finite number of options, which are all variants of a same domain. For example, if an accident at a reactor of a nuclear plant causes an explosion, the generic process model for handling nuclear disasters need to be adapted to this specific emergency. In this case, questions can be used to inquire about the characteristics of the emergency, e.g. the location, size and type of plant, the nature of the explosion, the number of people injured, etc. Via so-called *facts* that represent answers, the *questions* are linked to well-identified variation points in the process model for their configuration. Questions are expressed in natural language, thus can be answered by domain experts without extensive knowledge of the underlying process model. This abstraction from unnecessary information is particularly needed when the involved decisions are to be taken in a critical timeframe. The adaptation process is also facilitated via the use of *order dependencies* and *domain constraints*. The former provide fine-grained control over the order in which questions are presented to users; the latter capture interdependencies among facts which can be used to prevent the user from making inconsistent choices a priori. In this way we can first pose those questions that highly impact on the overall adaptation and (partly) infer the answer to subsequent questions, thus allowing the user to save time. In the presentation, we will show the use of an interactive questionnaire tool for the adaptation of an emergency-handling process that is defined as a configurable EPC extended with role assignment and object flow.