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 [1]. 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) [2, 3] 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 [2] 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 subtasks. 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
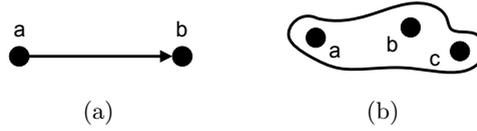
Fig. 3. (a) Graph-based control flow edge, (b) hypergraph-based control flow edge

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. At the first step of the process execution either activity a, b, or c can be performed. At 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 [4, 5]. 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 $\mathcal{P}(N) \backslash \emptyset$, 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 $e_4 = \{v_5, v_8\}$.
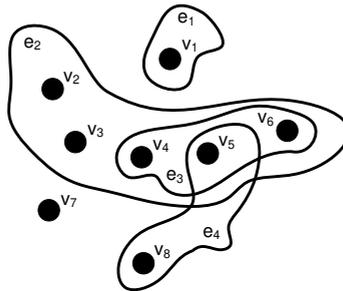

Fig. 4. Hypergraph example

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 $e = \langle I(e), O(e) \rangle \in E$, $A \cap E = \emptyset$*
  − *$I : E \to \mathcal{P}(A)$ is a function defining edge input activities*
  − *$O : E \to \mathcal{P}(A) \setminus \emptyset$ is a function defining edge output activities*
  − *$\forall e \in E : I(e) \cap O(e) = \emptyset$*
○ *T is an edge type function, $T : E \to \{and, xor, or\}$.*

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* $S : A \to \mathbb{N}_0 \times \mathbb{N}_0$ *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 [6].

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 $\epsilon^* : A \to \{true, false\}$.

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

The $cond : E \times A \to \{true, false\}$ 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 $\epsilon : A \times A \to \{true, false\}$.

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

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).
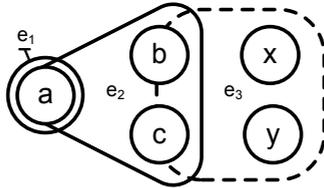

Fig. 5. Flexible process graph example

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, $T(e_1) = T(e_2) = and$, $T(e_3) = xor$. 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.
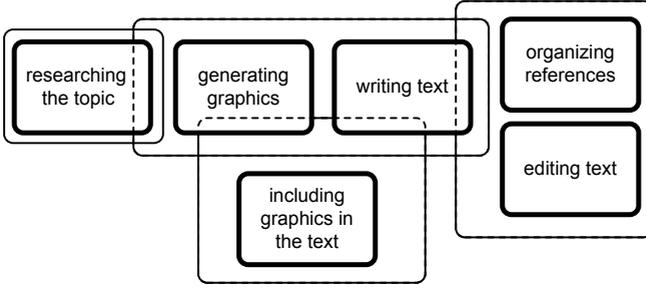
Fig. 6. FPG model of process from Figure 2

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 $T(e_1) = T(e_2) = T(e_3) = T(e_4) = and$.

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 activities. Then, the termination condition holds and the 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 [7]. Following, we discuss approaches that address the dimensions of change in process models.

ADEPT$_{flex}$ [8] 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 [9] 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 [10, 11] 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 [7] 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 [12]. Inspired by hypergraph formalism, the authors propose metagraph structure. As stated in [13], 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 [14, 15, 16]. Activities in a metagraph-based workflow are represented by arcs that relate objects consumed and produced during activity execution.

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