Flexible Process Graph: A Prologue

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. Businesses document their operational processes as process models. The common practice is to represent process models as directed graphs. The nodes of a process graph represent activities and directed edges constitute activity ordering constraints. A flexible process graph modeling approach proposes to generalize process graph structure to a hypergraph. Obtained process structure aims at formalization of ad-hoc process control flow. In this paper we discuss aspects relevant to concurrent execution of process activities in a collaborative manner organized as a flexible process graph. We provide a real world flexible process scenario to illustrate the approach.

Key words: business process modeling, hypergraph-structured process, ad-hoc process, flexible process

1 Introduction

In the dynamic and competitive business environment of nowadays it is essential for companies to introduce technologies that allow competitive market advantage. Products and services need to be served reliably, fast, at the best price and quality. Companies use process modeling techniques to document, study, and improve their operational procedures. 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]. Each business process model restricts a collection of process instances that cover particular business task handling. Every business task can be managed differently under special conditions, thus resulting in different process instances [2,3].

In many cases, an essential part of overall company success relies on running processes that assume ad-hoc nature. Such scenarios can not be solely described by the best-practice experience, but rely on process participants' creativity (process participants' flexible behavior). Alternatively, process flexibility can be explained by a dynamic process environment that influences process decisions and thus control flow routing. Process flexibility is also important in a highly collaborative environment where process participants collectively contribute to the achievement of a process goal. Such processes are characterized by numerous synchronization points and work handovers which might result in long process delays. Much research effort is invested in the study of the proposed scenarios. We concentrate on issues relevant to the task of process modeling and aim at full control over the execution of ad-hoc processes.

In [4] we introduced flexible process graph (FPG) as a formal approach for definition of ad-hoc process control flow. In this paper we discuss how the FPG formalism can be used to represent concurrent ad-hoc processes executed by different process participants.

The rest of the paper is organized as follows. In the next section we briefly sketch the FPG formalism. Afterwards, in section 3 we discuss issues relevant to the concurrent FPG activity execution by different process participants. Subsequently, section 4 illustrates presented concepts with a real world scenario of an ad-hoc business process and shows how FPG can be used to formalize it. Concluding remarks complete this paper.

2 Flexible Process Graph Foundations

In this section we briefly present the main concepts of FPG. FPG was first introduced in [4] and is a formal way for representing ad-hoc process control flow. In the core of FPG lies generalization of a directed process graph edge which defines a sequential execution of adjacent activities. In mathematics, generalization of a graph is a hypergraph [5,6]. Hypergraph edges (hyperedges) are arbitrary sets of nodes. Thus, a hyperedge is an edge that can connect multiple activities. As opposite to a graph-based sequence control flow pattern, it is allowed that within a hyperedge a process participant can choose which activity to execute next. A process model becomes hypergraph-, rather than graph-structured:

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*
  - – $O : E \to \mathcal{P}(A) \backslash \emptyset$ *is a function defining edge output activities*
  - – $\forall e \in E : I(e) \cap O(e) = \emptyset$
- ◦ T *is an edge type function,* $T : E \to \{and, xor, or\}$.

Each edge $e \in E$ in FPG is split into two subsets of input $I(e)$ and output $O(e)$ activities to obtain a *directed* hypergraph. Unlike regular graph-structured process models that contain special routing nodes—gateways, FPG introduces edge types that implement routing decisions. The structure of FPG is fixed and does not change during execution of a process instance. Dynamics of a process represented as FPG is specified by process state transitions:

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 a certain state, each activity node $a \in A$ 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.

Process Instantiation FPG process 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. An activity a is enabled at process start if $\epsilon^*(a)$ holds:

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

The *cond* 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 $a \in A$ 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 An activity $a \in A$ can be enabled after execution of an activity a_β if $\epsilon(a_\beta, a)$ holds:

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

An activity a enabling depends on execution of the preceding activity, e.g., a_β. An activity a can be enabled if there exists an edge $e \in E$, 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. Also, 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 From Formalism to Real World Business Processes

In this section we formally define the mechanism of activity assignment to different process participants and address the FPG activity concept as a time lasting phenomenon.

3.1 Process Roles

Activities in business processes are either automated by software systems or executed manually by people. Following, we discuss aspects concerning the assignment of process activities to agents that actually execute them. For the sake of simplicity we abstract from differentiating human and software agents and refer to them as roles. Each role is a sequential system, i.e., can be in the process of execution of only one activity at each moment in time. Therefore, concurrent activity execution can only be achieved by several roles executing different activities. Following, we formally define process role assignment:

Definition 3. *A flexible process graph $FPG = (A, E, T)$ role assignment is a pair (R, W) where:*

- ◦ *R is a finite set of roles*
- ◦ *$W : A \rightarrow \mathcal{P}(R) \backslash \emptyset$ is a roles assignment function.*

Each activity in FPG must have at least one role assigned. Each activity in FPG can be associated with several roles. Once enabled, an FPG activity $a \in A$ can only be executed by a role $r \in W(a)$.

During FPG process instance execution, each participating role can observe a subset of activities currently available for execution by the role—a role task list. By selecting and executing an activity from the proposed list the role contributes to the achievement of a process goal. The assignment of roles to FPG activities allows us to formally define a concept of a role task list.

Definition 4. *A role task list for the role $r \in R$ from the role assignment (R, W) for the flexible process graph $FPG = (A, E, T)$ is a function L, where:*

- ◦ *$L : R \rightarrow \mathcal{P}(A)$ is defined on a subset of FPG activities*
- ◦ *$L(r) = \{a \in A | r \in W(a) \wedge S_\omega(a) > 0\}$, where $r \in R$.*

Thus, a role task list is a subset of enabled activities of the FPG that are assigned to a certain role. Note, that a process participating role can consult on the number of enabled activity instances pending for execution by referring to the FPG state function S (cf. Definition 2).

3.2 Modeling Parallelism

So far we have presented FPG as the mechanism to define flexible activity enabling scenarios. Similar to Petri net [7] transitions, activity firing in FPG consumes no time. However, in real world scenarios instantiated business processes consist of activity instances that actually take time. Each activity instance is represented by its state transition system. Following, we discuss issues relevant to the interpretation of FPG when providing a structure to a process on a set of activities that can not be assumed instant by nature.

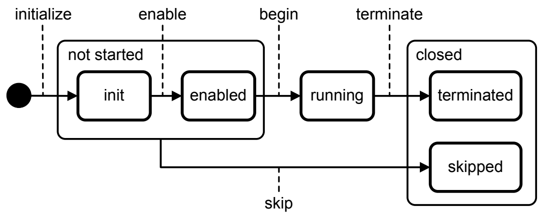

Fig. 1. Simple activity instance state transition diagram (adopted from [1])

Figure 1 shows a simple activity instance state transition diagram. When an activity instance is created it enters the *init* state. The enable state transition transfers the activity to the *enabled* state. Before an activity instance enters the *running* state it can still be *skipped* by the skip transition. An enabled activity instance can begin and enter the *running* state. Once accomplished, an activity instance enters the *terminated* state.

[1] also proposes a complex model of the activity instance state transition system. It allows an enabled activity instance to get *disabled* for some period of time. Also, a running activity instance can get *suspended* and afterwards return to the running state. Finally, the closed state in addition to terminated or skipped can also be *failed*, *undone*, or *cancelled*.

It is always possible and is allowed to come up with other state transition systems to represent activity instances. Therefore, instead of focusing on a one particular solution we rather state a list of generic requirements we expect any activity instance state transition system to fulfill if designed to be used as an FPG building block. An activity instance internal state transition system must contain the following generic states:

- *enabled* state—a state which means that the activity instance has to be accomplished in the process instance in order to realize the process goal
- *running* state—a state signals that work is currently conducted for the purpose of accomplishing the activity instance
- *terminated* state—a state which means that the activity instance was accomplished for the purpose of reaching the process goal.

Additionally, an activity instance state transition system must allow only a strict order on proposed activity states: first enabled, then running, and finally the terminated state. Once an activity is in one of the proposed states it can not return to the previous one given by the order. However, other states might be injected in between, e.g., an enabled activity can be disabled for some period of time or a running activity can be suspended and afterwards returned back to the running state.

The state transition diagram from Figure 1 satisfies the proposed requirements. It contains enabled, running, and terminated states and does not allow any scenarios that are forbidden by the proposed ordering constraints. Note, that one can decide on desired behavior by selecting appropriate mapping of generic states, e.g., one might decide to map the closed or the terminated state from Figure 1 onto the generic terminated state.

Once a direct correspondence between the generic activity instance states and the concrete activity implementation states is done, one can automatically map FPG transition states onto activity instance states. The generic enabled activity instance state corresponds to the FPG activity enabling and the generic terminated state corresponds to the FPG activity firing (cf. section 2). Thus, once an FPG activity is enabled following the FPG execution semantics a new activity instance should be transferred to the generic enabled state. Once an activity instance is terminated, has reached its generic terminated state, the corresponding FPG activity should fire to mark the FPG process state transition. The impact of the decision of a mapping between concrete and generic activity instance states should become clear now. In case we decide to map the generic terminated state onto the terminated state from Figure 1 the decision to skip the activity will not trigger the FPG state transition. Alternatively, if decided to accept the closed state as the generic terminated state, the FPG state transition will be triggered regardless of actually performing some work on accomplishing an activity instance or skipping it.

There is no direct mapping of the generic activity instance running state onto the FPG formalism. However, there is an additional constraint that no two activities executed by one role can be in the running state. We have already presented a concept of roles (cf. section 3.1). A role is a sequential system capable of executing assigned process activities. A role can consult its task list (cf. Definition 4) prior of selecting an activity for execution. Once started with the selected activity (entered the generic running state) the role should not be able to work on other activities. Only when the running state of the activity instance is left, the role can proceed with other activities.

In the simplest case it should be restricted that only one assigned role can execute an activity and that once started with the activity execution the role should accomplish it and bring it to the generic terminated state. However, more sophisticated scenarios can be envisioned. A role can suspend current activity execution in order to switch to another enabled activity and then return to the execution of the prior activity. Also, one can think of scenarios where several roles collaboratively accomplish an activity, i.e., several roles select the same activity for execution from their role task lists.

4 Flexible Business Process Scenario

In this section we present a real world flexible business process scenario. We show how this scenario can be formalized as a FPG.

The scenario describes a process of customizing, preparing, and shipping an order to a customer. The business process is decomposed to activities as follows. Once an order is confirmed by the customer, "*negotiate product customization*"

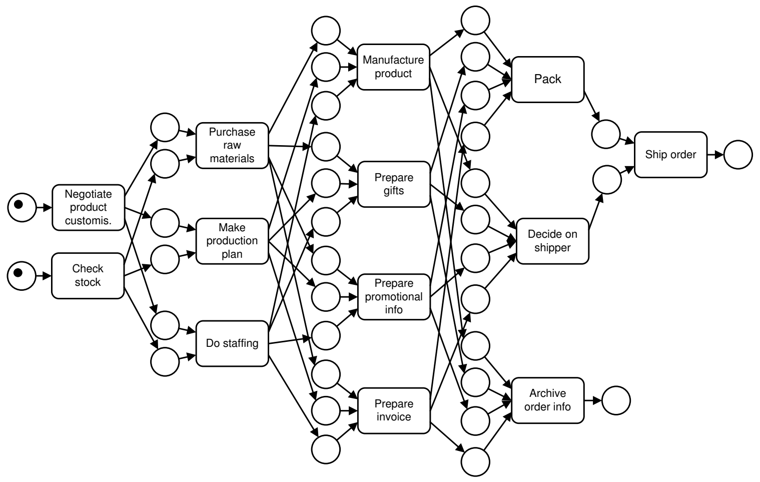

Fig. 2. Petri net model that captures flexible business process scenario

(NPC) activity takes place. In the scenario we do not concentrate on a specific product but assume a generic one, e.g., this can be a backpack. Following, "*check stock*" (CS) activity takes care of determining whether all basic materials are available to realize the order. One can "*purchase raw materials*" (PRM) required to customize a product, "*make production plan*" (MPP), and "*do staffing*" (DS) by assigning responsible for the task "*manufacture product*" (MP). Some additional work packets need to be performed prior of shipping the order to the customer; these are "*prepare gifts*" (PG) and "*prepare promotional info*" (PPI) to include into the order shipment. Also, somebody needs to take care and "*prepare invoice*" (PI). Once the order is ready someone has to "*decide on shipper*" (DOS), "*pack*" (P), and "*ship order*" (SO). Finally, it is required to "*archive order info*" (AOI).

It is clear that one might come up with several reasonable process models on the proposed set of activities. Following, we specify designed flexible activity execution constraints we assume for our scenario. A process instance can start with execution of either NPC or CS. Once both are accomplished, it is allowed to proceed in any order with execution of PRM, MPP, and DS activities. Once all the materials are available, production plan is ready, and workers are identified, it is possible to start with MP activity. At the same time somebody can take care of order supplements and perform PG, PPI, and PI activities. Once the ordered product is manufactured and all the supplements are prepared, activities concerned with order finalization can take place. It is required to accomplish AOI, DOS, and P activities. If order is packed and the delivery method is determined it is possible to proceed and do SO activity.

The Petri net model from Figure 2 captures the flexible process scenario. The model suffers from explosion of modeling constructs, in particular Petri net

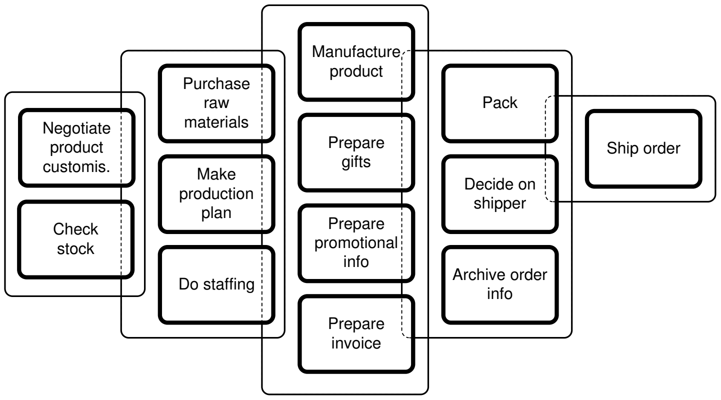

Fig. 3. FPG model that captures flexible business process scenario

places, that attempt when combined to represent all possible states of the flexible process scenario.

Figure 3 shows a graphical representation (for details refer to [4]) of the FPG model (A, E, T) that also captures our flexible process scenario, $E = \{e_1, e_2, e_3, e_4, e_5\}$ such that: $e_1 = \langle \emptyset, \{NPC, CS\} \rangle$, $e_2 = \langle \{NPC, CS\}, \{PRM, MPP, DS\} \rangle$, $e_3 = \langle \{PRM, MPP, DS\}, \{MP, PG, PPI, PI\} \rangle$, $e_4 = \langle \{MP, PG, PPI, PI\}, \{P, DOS, AOI\} \rangle$, $e_5 = \langle \{P, DOS\}, \{SO\} \rangle$, and function T is such that $T(e_1) = T(e_2) = T(e_3) = T(e_4) = T(e_5) = and$.

After the process initialization phase, activities NPC and CS get enabled, $S(NPC) = S(CS) = (1, 0)$. Eventually both activities are accomplished in the ad-hoc manner and result in the FPG state S such that $S(NPC) = S(CS) = (0, 1)$. Once in such a state further activities enabling takes place and $S(PRM) = S(MPP) = S(DS) = (1, 0)$. The process continues by obeying the FPG execution semantics until $S = (0, 1)$ for all the process activities. Then, the termination condition holds and the process terminates.

The amount of FPG model elements from Figure 3, contrary to the case of the model from Figure 2, clearly exhibits a linear behavior in respect to the amount of modeled execution constraints. One might introduce concurrency into the FPG model by distributing activities among different roles, e.g., supplementary activities PG, PPI, and PI can be assigned to a different role as MP activity.

5 Conclusions

The applicability of hypergraphs for process modeling task was investigated in [8,9,10,11]. The authors propose metagraph structure. Activities in a metagraph-based workflow are represented by arcs that relate objects consumed and produced during activity execution. Similar, the approach of case handling is strongly based on data as the typical product of the processes [12,13,14,15]. Case handling focuses on what can be done to achieve a process goal.

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. 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, and not based on data objects at hand.

This paper briefly presented the FPG formalism and contributed to the interpretation of FPG when modeled for concurrent execution of process activities by different process participants. This paper together with [4] finishes overall introduction of a novel approach for representing ad-hoc process control flow. The future work will be concerned with validation of the approach and study of its applicability in industry.

References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer Verlag (2007)
2. Davenport, T.: Process Innovation: Reengineering Work through Information Technology. Harvard Business School Press, Boston, MA, USA (1993)
3. Hammer, M., Champy, J.: Reengineering the Corporation: A Manifesto for Business Revolution. HarperBusiness (April 1994)
4. Polyvyanyy, A., Weske, M.: Hypergraph-based Modeling of Ad-Hoc Business Processes. In: Proceedings of the 1st International Workshop on Process Management for Highly Dynamic and Pervasive Scenarios, Milan, Italy (9 2008)
5. Berge, C.: Graphs and Hypergraphs. Elsevier Science Ltd. (1985)
6. Berge, C.: Hypergraphs: The Theory of Finite Sets. Amsterdam, Netherlands: North-Holland (1989)
7. Petri, C.: Kommunikation mit Automaten. PhD thesis, University of Bonn, Bonn, Germany (1962) (In German).
8. Basu, A., Blanning, R.: Metagraph Transformations and Workflow Management. In: HICSS '97: Proceedings of the 30th Hawaii International Conference on System Sciences, Washington, DC, USA, IEEE Computer Society (1997) 359
9. Basu, A., Blanning, R.: Metagraphs in Workflow Support Systems. Decis. Support Syst. **25**(3) (1999) 199–208
10. Basu, A., Blanning, R.: A Formal Approach to Workflow Analysis. Info. Sys. Research **11**(1) (2000) 17–36
11. 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
12. Aalst, W., Berens, P.: Beyond Workflow Management: Product-Driven Case Handling (2001)
13. Aalst, W., Weske, M., Grunbauer, D.: Case Handling: A New Paradigm for Business Process Support (2005)
14. Günther, C., Aalst, W.: Modeling the Case Handling Principles with Colored Petri Nets
15. Reijers, H., Rigter, J., Aalst, W.: The Case Handling Case (2003)