Semantic Analysis of Flow Patterns in Business Process Modeling

Pnina Soffer[1], Yair Wand[2], and Maya Kaner[3]

[1] University of Haifa, Carmel Mountain 31905, Haifa 31905, Israel
[2] Sauder School of Business, The University of British Columbia, Vancouver, Canada
[3] Ort Braude College, Karmiel 21982, Israel
spnina@is.haifa.ac.il, yair.wand@ubc.ca, kmaya@braude.ac.il

Abstract. Control flow elements are important in process models. Such elements usually appear in graphic models as splits and joins of activity sequences. Workflow patterns reflect possible executions of different configurations of splits and joins. However, despite the importance of process flow control and workflow patterns, no way exists yet to assure that a particular set of patterns is complete and non-redundant. We use an ontologically-based model of business processes to analyze the control configurations that can exist in a process model. A process is modeled in terms of state changes of the domain in which the process occurs. The state changes are controlled by laws which model the actions allowed in the domain. This model is notation-independent and enables incorporating goals into process analysis. We use the model to suggest classification of control configurations and identify configurations that assure the enacted process can always reach its goal.

1 Introduction

The possible flows in business process execution are determined by points where parallel or alternative process paths might be taken, or where such paths merge. This paper aims at systematically analyzing, defining, and distinguishing the different types of phenomena that are described by splitting and merging in business processes. The motivation for this work is threefold.

First, while splitting and merging structures in process modeling languages are frequently well-defined formally (e.g., [1][6]), they often do not convey a well-defined ontological meaning [8]. Second, splitting and merging structures are major sources of logical errors in process models (e.g., deadlocks and lack of synchronization [2][7][9]). Third, the available notation for splitting and merging is usually not expressive enough for representing and distinguishing the different cases of possible process behaviors. Usually, AND and XOR constructs are available, sometimes an OR too (e.g., EPC [10]). In a few cases (e.g., BPMN) there are specific constructs that can express more complicated behavioral patterns, a variety of which are depicted by workflow patterns [3]. In addition, split and merge of the same logical type typically have the same graphical notation. However, while this may provide for easy visual representation, in essence, splitting and joining stand for different real-world situations. Hence, this is a case of construct overload [13]. We believe this situation can lead to modeling errors. Hence, a clear distinction of the different situations represented by splitting and merging elements is expected to assist process designers in producing logically correct models.

In this paper we suggest real-world semantics to splitting and merging in process models, and a framework to enable a systematic analysis of splitting and merging configurations. Our analysis is based on the Generic Process Model (GPM). GPM is a notation-independent framework for analyzing business processes based on Bunge's ontology [4][5] and its adaptation to information systems [13][14].

2 The Generic Process Model (GPM)

This section provides an informal and brief presentation of the ontological state-based view of a process, which we employ for our analysis. The focus of analysis is a *domain*, which is a part of the world. A domain is represented by a set of *state variables*, each depicting the value of a relevant property of the domain at a given time. A successful process is a sequence of *unstable states* of the domain, leading to a *stable state*, which reflects the process *goal*. An unstable state is a state that must change due to actions within the domain (an *internal event*) while a stable state is a state that does not change unless forced to by action of the environment (an *external event*). Internal events are governed by *transformation (transition) laws* that define the allowed (or necessary) state transitions (events).

In these terms, the task of the process designer is to define the transition law (and ways to enact it) so that the process can accomplish its goal. The goal is a set of stable states on which the process must terminate. The law is specified as mappings between sets of states and is often defined formally by predicates over the state variables used to model the properties of the domain. The process goal may also be formalized in terms of predicates that specify situations that should be achieved by the process.

Process models usually include the concept of activity (a function, a task). The state, events and laws view of a process can be used to define activities. Consider a domain as comprising *sub-domains*, each represented by a subset of the domain state variables. The state changes of sub-domains are termed the *projections* of the domain's behavior on the sub-domains. A sub-domain is said to behave *independently* if its state changes are independent of the states of other sub-domains. We then say that the domain law projects a (well-defined) law on the sub-domain.

We view an activity as a change of a sub-domain from an unstable state to a stable state with respect to its (projected) law. As an independent sub-domain changes its state to a stable one, it is possible some other independent sub-domains will become unstable and will begin transforming. Thus, an activity can lead to other activities. As long as the process is active, at least one other sub-domain is still in an unstable state.

Since a process goal can be represented explicitly, the state-based supports an analysis of goal reachability. A process whose design ensures its goal will always be achieved under a given set events external to the domain (but affecting it) is termed valid [11][12]. Our analysis is intended to support the design of valid processes.

3 Modeling and Configuring Splits and Joins

3.1 Model Assumptions

First, we assume the designer defines the law to achieve the process' goal. Hence, we consider only valid process models, i.e. those that ensure goal reachability. Our analysis depends on the observation that for such models as long as the enacted process has not reached its goal, at least one sub-domain is unstable or may become unstable as a result of a time-related event.

Second, we assume that the granularity level of the model is defined in a manner that supports the business needs. In particular, a repeated activity (e.g. processing several replications of the same product) can be viewed as one activity. Hence, we do not address a flow of multiple instances in our model.

Third, our model does not incorporate durations or resources availability. Activities are enacted immediately when they are enabled.

Finally, for simplicity we only consider a binary splitting. The model can be readily extended to address cases of more than two sub-domains.

3.2 Characterizing Parameters of the Model

Under our basic assumptions we identify five parameters to characterize all splitting and merging situations. Using the requirement that the process should reach its goal, the combinations of possible values of these parameters will determine the set of acceptable combinations of splitting and merging configurations.

Parameter 1: Domain Decomposability
Splitting and merging can relate to one of two basic situations. First, a set of states achieved at a certain point in the process may be partitioned into two or more subsets and the next transformation is defined differently for each subset. Such partitioning might occur because the law at this state becomes “sensitive” to a certain state variable. Consider, for example, a process where a standard product is manufactured, and then packaged according to each customer's requirements. Until production is completed, the customer is not considered (even though this information may be known). At the completion point, the "customer" state variable determines which packaging action will be performed. This situation is clearly an XOR split, since the domain may take exactly one of the packaging paths available.

Second, there may be a point in the process, where the domain can be decomposed into two or more independently behaving sub-domains. In such cases, for the process to continue, at least one sub-domain must be unstable. Several possibilities exist. First, several sub-domains are always in unstable states, and will change concurrently (AND split). Second, any number of sub-domains (but at least one) can be unstable (OR split). Third, exactly one sub-domain should be in an unstable state and proceed to change (XOR split). In the process discussed above, once products are ready, two independent concurrently transforming sub-domains exist: one where shipment to the customer is arranged and one where products are transferred into the warehouse.

Our first characterizing parameter identifies whether the process domain is decomposable or not. The following four parameters apply only to decomposition-related splitting and merging.

Parameter 2: The Number of Paths
For a decomposable process domain three possibilities exist:

1. Both sub-domains are in an unstable state, thus they will transform in parallel. In this case the process has *only one path* (no selection decision is made). The "splitting" is merely a result of the decomposition. This situation is typically described by “AND” splitting elements in process models.
2. Depending on some state variable(s) value(s), exactly one sub-domain can be in an unstable state. Hence, an exclusive choice between *two possible paths* is made. This situation is typically described by “XOR” in process models.
3. Depending on some state variable(s) value(s), at least one or both sub-domains can be in an unstable state. The process has *three possible paths*: (1) one sub-domain is active, (2) the other is active, and (3) both are active.

Parameter 3: Past Awareness at the Merge
In standard process design, the merge condition reflects the type of preceding split. This entails an implicit assumption that the merge decision is “aware” of the process “history”. However, this cannot be taken for granted. We therefore incorporate a three-valued parameter to reflect the information available at the merge point.

1. No awareness – nothing is known about the preceding split. In other words, the view at the merge is purely *local*. We note this possibility is not of much interest, since usually the process designer is aware of the process structure.
2. Topology awareness – the type of split that precedes the merge is available at the merge point. This information can be considered available to the designer and hence incorporated in the law governing the behavior at the merge point.
3. Enactment awareness – this means that when the process is executed it is known at the merge point what happened at the preceding split. Specifically, for a two- or three- path split, it is known which path was actually chosen.

Parameter 4: Entry Condition into the Merge Sub-domain
The merge sub-domain is the sub-domain whose instability is affected by state variables of the two sub-domains. Several possibilities exist, but not all of them ensure a valid process. For example, each of the preceding sub-domains might activate the merge sub-domain, but not when both complete at the same time. It can be shown that only three cases exist for a valid process model:

1. Each branch is sufficient: stability (completion of action) of each preceding sub-domain causes instability of the merge sub-domain, independent of whether one or both sub-domains were activated.
2. Each branch is necessary and both together are sufficient: Only when both sub-domains reach stability (complete their activities) the merge sub-domain will become unstable. This is a *synchronizing merge*.

3. A specific branch is necessary and sufficient: stability of a specific one of the preceding sub-domains is necessary and sufficient for instability of the merge sub-domain. This is an *asymmetric synchronization*, where the merge can be activated by one sub-domain, or synchronize the two sub-domains, depending on which one has completed first.

Parameter 5: Process Goal Requirement

When two sub-domains become active at the split point, it may be sufficient that only one of them completes for activating the merge sub-domain, thus continuing the process. However, even if the process continues, it is possible that the goal depends on state variables that have to be set by actions in the other sub-domain. For example, one branch, necessary for the process to continue, deals with obtaining components for assembling a product. However, the process goal also includes securing means to deliver the product and this requires actions in the other sub-domain. Hence, we distinguish between two cases:

1. The process goal does not require that both sub-domains complete their activities. Of course, one must still complete for the process to continue.
2. The process goal is dependent on the completion of both sub-domains.

Table 1. Valid and invalid design possibilities

Past awareness	Split type	Merge entry condition		
		Each branch sufficient	Both necessary & together sufficient	Specific one sufficient
No awareness	One path (AND)	Always possible	it is not known a split happened	it is not known a split happened
	Two paths (XOR)	The only available option	it is not known two options exist	it is not known two options exist
	Three paths (OR)	The only available option	it is not known three options exist	it is not known three options exist
Topology awareness	One path (AND)	Always possible	It is known both branches must activate	It is known both branches must activate
	Two path (XOR)	Always possible	Only one branch activates	It is known one branch activated – not which one
	Three path (OR)	Always possible	It is not known if two branches activated	It is not known which branch activated
Enactment awareness	One path (AND)	Always possible	Same as for topology awareness	Same as for topology awareness
	Two path (XOR)	Always possible	Only one branch activates	whichever branch taken should suffice
	Three path (OR)	Always possible	If known that both branches active – condition on both possible(*)	If known that both branches active – condition on each possible(*)

(*) The designer can specify conditions related to both branches, but should also allow for activating the merge domain on each branch if only one is activated.

Table 2. Pattern composition (NA– “not applicable”.)

No. of sub-domains	Number of paths	Past awareness	Merge entry condition	Goal requirement	Description	Example
1	2	NA	NA	NA	Single domain exclusive choice	Quality inspection determines whether a product is good and can be supplied (path 1) or needs rework (path 2). Merge after rework.
2	1	All values	One sufficient	Requires one domain	Concurrency with competition	An urgent loan is requested from two banks (each bank request is a domain). The first one to approve the loan is chosen.
2	1	Topology / enactment	One sufficient	Requires both	Concurrency with First-in-first-out (FIFO) merge	Sales data (domain1) and production data (domain 2) are collected for a periodical report, whose preparation starts once the first type of data arrives, but requires both to end.
2	1	Topology / enactment	Both necessary	Requires both	Concurrency with synchronization	When an order is received, the customer's credit (domain 1) and the availability of products (domain 2) are checked. After both complete, the order may be accepted.
2	1	Topology / enactment	Specific one sufficient	Requires one domain	Concurrency with asymmetric synch. / competition	When considering an architectural design, a professional drawing may be prepared (domain1); until it is ready, a draft may be made for demonstration purposes (domain 2). When the drawing is ready the draft is discarded and the process may proceed.
2	1	Topology / enactment	Specific one sufficient	Requires both	Concurrency with asymmetric synch. / FIFO	Information about product requirements (domain 1) and production resources (domain 2) is needed for production planning, which can start when product requirements are known, even if the resources are not known yet. When resource information arrives planning can complete.
2	2	All values	Either one sufficient	Requires one domain	Two domain exclusive choice	A product can be manufactured (domain 1) or outsourced (domain 2). Goal includes having product.
2	3	All values	One sufficient	Requires one domain	Multi-choice with competition	A message can be sent by mail (domain 1) or by fax (domain 2). If it is sent by both, the first one that arrives is addressed, and the other one is discarded.
2	3	Enactment	One sufficient	Requires both	Multi-choice with First-in-first-out (FIFO) merge	A person makes a claim to the insurance company after a car accident regarding car injury or physical injury (or both). Each claim (domain) is processed separately, and when its processing is completed the person is paid.
2	3	Enactment	Both necessary	Requires both	Multi-choice with synchronization	Planning a trip may involve flight booking (domain 1) and hotel reservation (domain 2). If both are performed, they have to be completed before the trip can take place.
2	3	Enactment	Specific one sufficient	Requires one domain	Multi-choice with asymmetric synch. / competition	A new employee receives salary only after his details are recorded in the information system (domain1). Before that, the company may make a cash advance (domain 2), not needed if salary is paid on time.
2	3	Enactment	Specific one sufficient	Requires both	Multi-choice with asymmetric synchronization / FIFO	Planning a trip may involve flight booking (domain 1) and hotel reservation (domain 2). Trip can begin before all hotels are reserved, but not before flight is booked. Some hotels can still be booked during the trip.

3.3 Combining Parameter Values

The combinations of the above parameters provide possible split and merge configurations. Analysis of these configurations can identify those that will always progress and those that may fail to progress in certain situations, thus preventing a process from reaching its goal. The latter should not be used in valid process models.

Table 1 presents the possible combinations of parameters 2-4. Combinations allowing goal reachability are marked by clear boxes. Combinations that do not guarantee process success are marked by shaded boxes. For example, a two-path split (XOR) leading to a merge where both branches are necessary cannot progress to the process goal, and is hence not a valid configuration.

Table 2 enumerates all possible valid combinations of the five parameters. In some cases, as indicated in the table, different values of the same parameters support the same behavior (e.g., when one sub-domain is sufficient for activating the merge, all valid values of past awareness may be considered equivalent). To illustrate the derivation of Table 2, consider, for example, lines 2 and 3. Line 2 refers to the case where the process continues when one branch completes, regardless of whether a second branch even exists. Hence, the goal should be reachable based on any of the branches completing. On the other hand, in line 3 it is known two branches were activated. Hence the process goal can depend on both.

4 Conclusion

Attempts to distinguish different types and behaviors represented by splitting and merging elements in process models were made in the past. The most comprehensive one is probably the workflow pattern initiative [3]. Workflow patterns address, in addition to flow structures, workflow management system functionality (e.g., cancellation). Some of our patterns are included in the control flow patterns, while others are not. Specifically, we distinguish between single-domain and two-domain XOR, and identify asymmetric synchronization, where synchronization may or may not be required, depending on the branch which completes first.

This paper adds to extant analysis in several ways. First, it anchors splitting and merging elements in an ontological theory, thus suggesting a real-world interpretation of process control elements. Second, it provides a framework for systematic identification of splitting and merging configurations. The framework is based on an explicitly specified set of assumptions and parameters. It can be shown that under this set the identified set of patterns is complete, if a process model is required to assure that the process can always reach its goal.

The framework thus forms a basis for further systematic analysis that can be achieved by relaxing these assumptions. Such analysis can yield a broader set of patterns, whose completeness with respect to its set of underlying assumptions can be analyzed. Third, the identified patterns include cases which have not been indicated and discussed so far. Finally, we identify patterns that provide for goal reachability of the designed process, thus suggesting a way to support the task of process designers.

Future research should investigate the applicability of the identified patterns as a benchmark for evaluating and developing process modeling languages and as guidance to the actual practice of process design. We believe that incorporating the view suggested in this paper into the practice of modeling (through, e.g., modeling rules) may lead to an improved quality of designed processes.

Acknowledgement. This work was supported in part by a grant to one of the authors from the Natural Sciences and Engineering Research Council of Canada.

References

[1] van der Aalst, W.M.P.: The Application of Petri-nets to Workflow Management. Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)

[2] van der Aalst, W.M.P., Hirnschall, A., Verbeek, H.M.W.: An Alternative Way to Analyze Workflow Graphs. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 535–552. Springer, Heidelberg (2002)

[3] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)

[4] Bunge, M.: Treatise on Basic Philosophy: Ontology I: The Furniture of the World, vol. 3, Reidel, Boston (1977)

[5] Bunge, M.: Treatise on Basic Philosophy: Ontology II: A World of Systems, vol. 4, Reidel, Boston (1979)

[6] Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of control flow in workflows. Acta Informatica 39(3), 143–209 (2003)

[7] Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 82–97. Springer, Heidelberg (2004)

[8] Rosemann, M., Recker, J., Indulska, M., Green, P.: A Study of the Evolution of the Representational Capabilities of Process Modeling Grammars. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 447–461. Springer, Heidelberg (2006)

[9] Sadiq, W., Orlowska, M.E.: On Correctness Issues in Conceptual Modeling of Workflows. In: Proceedings of the 5th European Conference on Information Systems, Cork, Ireland, pp. 943–964 (1997)

[10] Scheer, A.W.: ARIS-Business Process Modeling. Springer, Berlin (1998)

[11] Soffer, P., Wand, Y.: Goal-driven Analysis of Process Model Validity. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 521–535. Springer, Heidelberg (2004)

[12] Soffer, P., Wand, Y.: Goal-Driven Multi-Process Analysis, Journal of the Association of Information Systems (forthcoming, 2007)

[13] Wand, Y., Weber, R.: On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. Journal of Information Systems 3, 217–237 (1993)

[14] Wand, Y., Weber, R.: Towards a Theory of Deep Structure of Information Systems. Journal of Information Systems 5(3), 203–223 (1995)