# Conceptualizing Routing Decisions in Business Processes: Theoretical Analysis and Empirical Testing


## *Abstract*

*Business process models are widely used for purposes such as information systems analysis, improving operational efficiency, modeling supply-chains, and business process reengineering. A critical aspect of process representation involves a choice among alternative or parallel routes. Such choices are usually represented in process models by routing structures that appear as "split" and "merge" nodes. However, evidence indicates that modelers face difficulties representing routing options correctly. Clearly, errors in representing routing options might negatively affect the effective use of business process models.*

*We suggest that this difficulty can be mitigated by providing process modelers with a catalog of routing possibilities described in terms that are meaningful to analysts. Based on theoretical considerations, we develop such a catalog and demonstrate that its entries have business meaning and that it is complete with respect to a defined scope of process behaviors that do not depend on resources or on software features. The catalog includes some routing cases not previously recognized. We tested experimentally the catalog in helping subjects understand process behavior. The findings demonstrate that the catalog helps modelers understand and conceptualize process behavior and that the likely reasons are its completeness and the practical terms used to describe its entries. .*



**Key words:**
Business Process Modeling, Routing structures, Cognitive aspects of modeling


## Introduction

Business process models play a key role in analyzing and designing business processes, implementing workflow systems, and analyzing information systems. Such models comprise two main types of elements; *activities*, and *routing* or *control flow* structures. Different flows can occur depending on conditions that arise during process execution. Routing components that appear as "split" and "merge" nodes in process models, indicate *possible* execution routes but not the actual routes that will be determined only when the process is executed. Hence, these elements are *abstractions* of possible business process flow decisions. In a split, process execution can take at least one of several routes. A merge node represents a point where the process continues in the same way for all routes possible at a previous

split point. Routing structures are included in most process modeling notations[1]. They endow process models with their richness in terms of execution possibilities and hence contribute substantially to the expressiveness of process modeling techniques. It is not surprising therefore that much work has explored routing aspects of process models. This includes developing techniques to represent process behavior, analyzing characteristics of process behavior (e.g. soundness, which is the ability to reach a definitive final state (Aalst 1997)), and conducting empirical studies on how routing structures affect the creation and use of process models (Mendling et al. 2006, 2007, 2008). A well-known project is the *Workflow Patterns Initiative,* which has sought to identify and define "patterns describing the control-flow perspective of workflow systems" (Russell et al. 2006).

Process routing elements pose both theoretical and practical challenges. First, it is unclear what constitutes a "good" set of routing behaviors. Most process modeling languages employ the basic constructs of XOR and AND, which can be used to model complex situations (Aalst et al., 2003). However, some languages have constructs to model more complex situations such as OR (EPC (Aalst 1999), YAWL (Aalst and Hofstede, 2005), and BPMN (OMG, 2006)), and Complex Gateway (BPMN). The Workflow Patterns collection (Russell et al., 2006) includes other complex behaviors (e.g., Discriminator).

Second, practical difficulties exist in modeling routing situations in realistic business settings. Some practical situations cannot be readily modeled. For example, assume that a component is urgently needed. The component is both ordered from a supplier and assembled locally. If the order arrives first, local assembly can be stopped. If local assembly is completed first, the process continues but the order will still arrive. In most modeling languages, this behavior, although plausible, might require a significant combination of constructs to be represented. In contrast, model fragments that do not clearly map to practical situations can be constructed.

Third, empirical research indicates that routing decisions cause difficulties in creating and understanding process models. For example, Mendling et al. (2006, 2008) found a positive correlation between the number of control nodes and the number of errors in a process model. Other studies (e.g., Mendling et al. 2007)

[1] An exception is the domain-specific language Picture (Becker et al., 2010). Picture uses building blocks where decisions are embedded in activities (without splits). This is aimed at avoiding multiple conflicting representations of a situation, and to facilitate automated model analysis.

found a negative correlation between the number and degree (number of related paths) of control nodes and the understandability of models.

A likely cause of difficulty in creating and understanding models is the need to conceptualize complex business process behavior. Routing points reflect *abstractions of many possible process executions*. Thus, creating a process model requires abstracting actual situations. Understanding process models requires translating model constructs to abstract concepts reflecting alternative behaviors.

In this work we propose that conceptualizing process behavior can be facilitated by using a *classification* scheme of routing phenomena, defined in terms that reflect *what an analyst can observe and recognize* in a business domain. We develop such a classification in a *catalog* of fundamental process routing situations ('split' and 'merge' types). In the catalog, each item is assigned a clear business meaning, implementing a specific and unique business rule. We limit the phenomena we classify to those that reflect fundamental business process routing decisions, and exclude their implementation aspects (such as resources available and software mechanisms). We show that for binary split and merge cases our catalog is complete and non-redundant.

We propose that when identifying a situation as an instance of a specific class, the analyst can better understand and explore it by inferring additional information about the situation (Parsons and Wand, 2008). This understanding can help an analyst choose the correct representation in business process models.

In terms of design science (Hevner et al., 2004), we develop an artifact–the specific catalog–by applying a theoretical view of a process as state changes in the (business) domain. We then evaluate the catalog in three ways. First, we show by theoretical analysis that it is complete (for its intended scope) and non-redundant. Second, we demonstrate via business examples that the entries (categories) are meaningful and have business relevance. Third, we conduct two experimental studies that provide evidence to the usability and usefulness of the catalog in conceptualizing business behavior.

In the following, we first apply a cognitive problem-solving perspective to the difficulties encountered in process modeling. We then use a formal view of business processes as state changes to analyze process routing decisions and develop the catalog of split and merge possibilities. Following the analysis, we evaluate the catalog and describe two studies to test how well it can support conceptualization of

process behavior. We then discuss the significance of the results, summarize the work, and suggest further research.

## Cognitive Aspects of Process Modeling

Empirical observations (Pinggera et al., 2012) have indicated that process modeling involves three phases. The first is comprehension, in which the modeler develops an understanding of the represented domain. The second, modeling, occurs when this understanding is transformed into modeling constructs. Thirdly, reconciliation occurs when model elements are reconciled, moved, and renamed, to improve appearance and clarity. These three phases are repeated, each iteration relating to a chunk of the model. Iterative chunking has been addressed by cognitive problem-solving theories (Newell and Simon, 1972), which suggest that tasks are typically addressed in smaller parts (chunks) due to limitations in working memory (Miller, 1956).

We consider the construction of a process model to represent a given domain behavior as a problem-solving task, and the model as the solution. We focus on the comprehension phase, when the modeler develops a domain understanding. The domain understanding will be mapped into constructs of a modeling language.

According to Newell and Simon (1972), the problem-solver formulates a mental model of the problem, and uses it to reason about the solution and to apply solution procedures. The importance of the mental model has been widely recognized. For example, Jonassen (2000) claims that it is the mental construction of the problem space that is the most critical for problem solving. Simon (1981, p. 153) claims that "solving a problem simply means representing it so as to make the solution transparent." Savelsbergh et al. (1998) discuss possible functions served by the mental model in understanding and solving problems.

In process modeling, solution procedures (in the sense of Newell and Simon (1972)) entail mapping the mental model of the process domain behavior into a particular modeling language. According to Newell and Simon (1972), the mental model is affected by the characteristics of the task, and the methods used to achieve it. For example, if the intended model is a Petri net, the mental model would most likely relate to tokens and their dynamics. Similarly, Larkin (1985) considers that a mental problem representation means classifying its description into a schema of concepts to which solution procedures can be applied.

In the cognitive schema theory (Derry, 1996), mental models are cognitive schemas that help to understand a specific situation and solve the related current problem. Mental models are constructed by using lower-level cognitive schemas, called *memory objects*, as building blocks. Memory objects are components of human knowledge stored in long-term memory. The simplest objects are basic concepts, called *p-prims*. Above them are *integrated objects* that enable people to recognize and classify patterns in the external world, so they can respond with appropriate mental or physical actions. A mental model is constructed by mapping memory objects onto components of a real and currently faced phenomenon, reorganizing them, and connecting them to form a model of the whole situation.

Construction of the mental model is highly affected by the availability of memory objects. According to the cognitive load theory (Chandler and Sweller, 1991), the burden on the limited capacity of working memory can be reduced by using schemas that allow categorizing multiple elements as a single element (Paas et al., 2003).[2] When the cognitive schemas used are low level and require further integration to construct a mental model, cognitive load is increased. This might reduce efficiency in performing the task (Paas et al., 2004).

Consider the use of process modeling constructs as memory objects to construct mental models of process phenomena. Actors and activities are concrete and observable elements. However, routing elements are *abstractions* of multiple possible occurrences of the process. Hence, recognizing and classifying these elements is more difficult than it is for concrete aspects such as actors or activities.

The abstract nature of routing elements is manifested in attempts to analyze process modeling languages in ontological terms. Rosemann et al. (2006) used Bunge's ontology, while Santos et al. (2010) applied the UFO ontology. Neither found a direct mapping of split and merge constructs to ontological concepts (an indirect link was suggested by Santos et al. (2010)). These outcomes indicate difficulties in relating routing elements to "real world" phenomena.

The constructs of process modeling languages might serve as memory objects in the mental model of a modeler[3]. However, the meaning of such constructs is often inaccurately defined. For example, Dijkman et al. (2008) observe that BPMN 1.0 notation has ambiguities. They claim that syntactic rules are comprehensively

[2] For cognitive load considerations in process modeling, see, for example, Figl et al. (2010).
[3] Even when using process modeling constructs, a mental representation is not a complete and ready process model, as it does not include syntactic and layout aspects.

documented throughout the BPMN standard specification, but the actual semantics are described in only a narrative form whose terminology is occasionally inconsistent. Rittgen (1999) argues that an OR-join in EPC may have different possible interpretations[4]. Several other studies observed and attempted to solve problems that may arise in EPC due to the lack of clear semantics of OR merge (Aalst, 1999; Kindler, 2006; Mendling and Aalst, 2007). In summary, the meaning of routing-related modeling constructs is often ambiguous. This can hinder the use of language constructs as memory objects to support recognition and classification of domain phenomena. Furthermore, modeling languages offer a limited set of concepts (typically AND, XOR, OR split and merge nodes) that might require still more integration to represent the full variety of routing behaviors.

To overcome the imprecise meaning of process modeling language constructs, attempts were made to use Petri net concepts (e.g., Mendling and Aalst, 2007; Dijkman et al., 2008). Petri nets describe dynamics in terms of creation and destruction of tokens (Petri, 1962). Token-based representation complements process models by enabling precise reasoning about dynamics. However, as tokens are primitive concepts they would need to be integrated with other modeling constructs, possibly leading to increased cognitive load. Moreover, more than one translation of a token-based representation into a process model might exist (Vanderfeesten et al., 2008) leading to further translation problems.

To summarize, the concepts of process modeling languages do not appear to provide an appropriate set of memory objects in mental models of process behavior. This motivates us to develop a catalog of routing behaviors to fill this gap. To provide the required support, the catalog should (a) use terms that can be readily related to domain ("real world") behavior, (b) use precisely defined concepts, and (c) include a complete set of possible routing behaviors, to avoid or minimize a need for integration. To reduce cognitive effort, it would also be desirable to reduce redundancies in the catalog.

## Defining Business Processes

We analyze routing elements in process models in terms of *observable behavior of the domain* in which the process takes place. We focus on the *semantics* of process models, not on formalization in terms of modeling constructs. Specifically, we seek

[4] Wait for all to activate, activate on the first that comes, and activate every time.

all generic cases of possible process routing. Such choices typically appear as *split points* and– following splits–*merge points* in process models.

We analyze cases where actual domain behavior might take more than one course of action *as perceived by process stakeholders*. We do not include workflow software mechanisms (e.g. interrupt features), resource considerations (where process execution is affected by resource availability), or whether or not several activity instances can be executed concurrently (also related to resources). We consider multiple concurrent occurrences of the same case type as one generic behavior, and will seek completeness with respect to this scope.

We employ the Generic Process Model (GPM) (Soffer and Wand, 2004; 2007), which employs ontological concepts. GPM defines an enacted process as a *set of state transitions in the process domain*. Transitions result either from transformations within the domain (internal domain dynamics), or from effects of the domain environment (external events). A process ends when the domain reaches a state in a *goal* – a set of states desirable to stakeholders where no more changes can occur due to internal domain dynamics.

The ontological concepts underlying GPM are taken from Bunge's work (Bunge, 1977; 1979) and its adaptations to information systems (Wand and Weber, 1990; 1995) and business process modeling (Soffer and Wand, 2004; 2007). For our purpose, representing a process this way has three advantages. First, it is *linked to domain phenomena* by its ontological roots. Second, it is not confounded by process modeling notation. Third, it has been used for analyzing process behavior, especially whether the process can meet its goals (Soffer and Wand, 2005).

Empirical evidence exists to show that Bunge's ontology is applicable when evaluating business process techniques. Recker et al. (2011) found that ontology-based predictions about BPMN deficiencies were corroborated by practitioners. However, whether or not the use of GPM concepts can provide a useful catalog of process behaviors requires empirical corroboration. We therefore include in this work two empirical studies that test whether our analysis has useful outcomes.

We start by introducing the GPM view of process. We then use this view to identify generic routing situations (for both split and merge possibilities).

### *Ontological Concepts*

The ontological framework considers a world made of *things* that possess *properties*. A property can be *intrinsic* to a thing (e.g. assets of a company) or *mutual* to several things (e.g. salary paid to an employee by a company). Things can combine to form *composites* that have *emergent properties* that arise due to interactions among the components and are not properties of the individual components (e.g. the processing power of a computer). Properties are perceived by humans as *attributes* whose values are functions of time. A *functional schema* is a set of attribute functions that represent a view of similar things. This view reflects the ***purpose*** of an observer. The *state* of a thing at a given time is the set of values of the attribute functions. We refer to these functions as state variables.[5]

In our analysis, we model the process domain or parts of it (sub-domains). It is particularly important that the choice of state variables reflects the stakeholder's[6] view of the domain and sub-domains. Specifically, observers may vary in the ***granularity*** (level of detail) they use when defining the state. Different states for one observer might be considered as one state by another observer. For example, when a product has been manufactured, one observer may differentiate several states, depending on how many components are still available. Another observer may only have one end state, i.e., the product has been assembled.

When the properties of a thing change, the change of state is an *event.* States are considered discrete in time, and thus events occur at well-defined points in time. Events can occur either due to *internal transformations* (internal events) in the thing, or due to its *interactions* with other things (external events). Since the event concept is based on a chosen functional schema, one observer may perceive an event, while another observer does not. Events that appear different to one observer may also be considered as one event by another observer. For example, one observer may perceive arrival of any order as one type of event, while another observer will consider orders from different types of customers as different types of events.

---

[5] Formally: A functional schema is a model of a thing x: $x_m = (M, \underline{\boldsymbol{F}})$, where $\underline{\boldsymbol{F}}$ is a set of functions from a domain M to a set of co-domains: $\underline{\boldsymbol{F}} = \langle F_1,\ldots,F_n\rangle$: $M \rightarrow V_1\times\ldots\times V_n$ . $F_i$, $1 \le i \le n$, represents a property of X and is called the i-th state function (variable) of x. $S(X) = \{\langle x_1,\ldots,x_n\rangle\} \in \{V_1\times\ldots\times V_n \mid x_i = Fi(M)\}$ is the possible state space of X. [Wand & Weber 1990]

[6] We recognize that there may be several stakeholders whose view may matter for a process. Here we assume that the model reflects the view of a single "process owner".

We denote the conceivable states (conceivable combinations of state variables) of a thing x by S(x). Not all conceivable states can materialize. The constraints defining allowed states are termed *state laws* and the states that conform to these laws are termed *the lawful states* of a thing x, denoted by $S^L(x) \subseteq S(x)$. Similarly, not all conceivable internal events (transformations in the thing) can occur. We term the mappings that determine the possible events that can occur in a given state due to transformations in the thing the *transition law*. Formally:

$L: S^L \rightarrow \mathcal{P}(S^L)$ (where $\mathcal{P}(S^L)$ is the power set of $S^L$). $\forall\, s \in S^L \;\; L(s) \in \mathcal{P}(S^L)$.

We distinguish three possibilities:

(1) $s \notin L(s)$. An internal transformation will occur and we term s an *unstable state*.

(2) $s \in L(s)$. An internal transformation may or may not occur.

(3) $s \in L(s)$ and L(s) is a singleton: $|L(s)|=1$. $L(s)=\{s\}$ means that no internal transformation can change it. We term such a state *stable*.

A stable state of a thing can only be changed by actions of other things. Such changes are termed *external events*. Conversely, the occurrence of a state change without an action by another thing indicates that the state prior was not stable.

An internal event is represented by <s,L(s))>. An external event is represented by <s,e⊗s> where e⊗s is the state resulting from an external occurrence, e, at s.

The purpose of observing a process state is to decide on actions. We consider states as *equivalent* if they lead to the same choice of action by a stakeholder. For example, it might be sufficient to know that when inventory drops below a certain threshold, a given action (e.g. replenishment) must be taken, while the (more detailed) state information includes the actual inventory level. Accordingly:

**Definition 1** ***(equivalent states)*****:** A set of states, $S' \in P(S^L)$ will be termed *equivalent* for a stakeholder if this stakeholder chooses the same action for each of these states.

**Formally**: Let $\mathcal{A}=\{a\}$ be a possible set of state-changing actions $a: S^L \rightarrow \mathcal{P}(S^L)$ (namely: $\forall a \in \mathcal{A}\ \forall s \in S^L,\ a(s) \subseteq S^L$) and $\mathcal{D}$ a decision function: $\mathcal{D}: S^L \rightarrow \mathcal{A}$. $S' \in \mathcal{P}(S^L)$ is an equivalence set (with respect to $\mathcal{A}$ and $\mathcal{D}$) iff $\mathcal{D}(s_1)=\mathcal{D}(s_2)\ \forall s_1, s_2 \in S'$.

Definition 1 interprets the abstract ontological notion of a transition law in terms of decisions about actions available to a stakeholder: $\forall s \in S^L$ if $\mathcal{D}(s)=a$, then $a(s)=L(s)$. For equivalent states, the same action will take place. We note that:

(1) In our formalization, the ***role of a stakeholder*** is abstracted to the decisions about actions. The notion of equivalence set of states reflects a stakeholder's view that these states are indistinguishable for deciding on the next action.
(2) Following (1), a repeating behavior is actually re-entering a set of equivalent states. There are two possible decisions: repeat the behavior or stop repeating. States in which the same decision is taken constitute an "equivalence set," although they may differ in additional state variables, and thus appear to be different.
(3) This view does not imply that the decision maker actually takes the actions.
(4) The actions $\mathcal{A}$ may include "no action."

To decide on actions, stakeholders need an assurance that the outcome will not be random. Hence, we introduce the idea of *predictable behavior*:

**Definition 2** ***(predictable behavior)***: A process domain (or sub-domain) has a ***predictable behavior*** for a subset of states $S' \subseteq S^L$ iff for each $s \in S'$ L(s) is an equivalent set of states.

**Formally**: Given $\mathcal{A}$ and $\mathcal{D}$, $\forall s \in S'$ $\forall s_1, s_2 \in L(s)$ $\mathcal{D}(s_1)=\mathcal{D}(s_2)$. This formal form is implied by the condition (see Definition 1) that $\forall s \in S^L$ if $\mathcal{D}(s)=\boldsymbol{a}$, then $\boldsymbol{a}(s)=L(s)$.

Definition 2 specifies that for predictable behaviors of each state in S', the decision will lead to equivalent outcomes. For the stakeholder, they will be indistinguishable regarding further actions that may be taken.

The above definitions lead to the following conclusion:

**Corollary**: Let $s \in L(s)$ where L(s) are equivalent states for the stakeholder (but not a singleton). From a stakeholder's point of view, the states will be considered stable.

Equivalence and hence stability depend on the stakeholder. This enables us to model cases where one stakeholder might take an action while another will not.

### *Process Domain and Sub-domains*

The properties and dynamics of the process domain reflect the properties, internal transformations, and interactions of its components. The composition of the domain determines which changes will be considered internal and hence governed by the process (reflecting stakeholder's decisions) and which changes will be considered external (and not controlled within the process). As explained above, we abstract the domain and its dynamics in terms of *relevant state variables* and the *changes* that can occur to their values. We assume that choosing state variables reflects the business semantics and the information relevant to stakeholders.

Process activities cause state transformations in the domain and typically might impact only parts of the process domain (such as organizational actors or units). We formalize such parts using the notion of *sub-domain* – a part of the domain represented by a subset of the state variables of the domain:

**Definition 3** ***(domain, sub-domain model)***: A model of *domain D* is a set of state variables $X^D$. A *sub-domain* Z is a part of the domain modeled by a subset of the domain state variables $X^Z \subset X^D$.[7]

Consider, for example, a business that assembles products based on customers' orders and then ships the products to customers. The domain D will represent the whole business. Examples for sub-domains are *assembly* (where state variables may represent status of work stations) and *shipping* (where state variables may represent orders ready to ship). The model of the domain D can also include emerging state variables, such as the overall status of a customer order.

The idea of sub-domain is similar to *workflow boundaries* used in Product-Based Workflow Design (Reijers et al. 2003), and to *artifacts* used in analyzing work processes. For example, Limonad et al. (2012) use "…Business Entities (BE) (a.k.a. Business Artifacts)… to conceptualize the organizational domain."

It is possible that for some domain states the internal transitions of a sub-domain depend only on its own state, and not on the rest of the domain. In such cases, the sub-domain behavior will be *predictable*. To explore such possibilities we first define the relationships between domain states and sub-domain states:

**Definition 4** ***(state projection, projecting set)*****:** The *projection* of domain state, s, on sub-domain Z (denoted $s_{/Z}$**),** is the set of values of $X^Z \subset X^D$ in s. The *projecting set* for a sub-domain state t (denoted $S^D(t)$) is the set of all domain (D) states that project to the sub-domain (Z) state t.[8]

A domain state projects onto a single sub-domain state. In contrast, more than one domain state will usually project onto a given sub-domain state. For example,

[7] Formally: A *domain model* is a set of state functions $D=\{f_1(t)\ldots f_n(t)\}$ and the state of the domain at a given time t is $x_k,=\{f_k(t), k=1,\ldots n\}$. The domain is modelled by $X^D=\{x_k; k=1\ldots n\}$ where the state of the domain at a given time is $s(D)=<x_1,\ldots x_n>$ (or simply s). A *sub-domain* is a part of the domain described by a subset of the domain state variables $X^Z \subset X^D$.

[8] Formally: Let $s(D)=<x_1,\ldots x_n>$ be the state of D and the state of domain Z such that $X^Z \subset X^D$ be $\{y_1,\ldots,y_m\}$ $(m<n)$. The projection $s_{/Z}$ on a state of Z is such that $y_k=x_j$, where $k=1,\ldots,m$ and $j \in \{1\ldots n\}$ and each j is counted at most once. The *projecting set* in S(D) for state v in sub-domain Z is the set $S^D(v)=\{s \in S(D) \mid s_{/Z}=v\}$.

"materials arrived from a supplier" and "materials arrived for processing from a customer," will project to one warehouse state, "materials available for storage."

Consider how a sub-domain changes its state when the domain changes its state. The domain states before and after the change project onto sub-domain states. If these projections are not equivalent states of the sub-domain for a stakeholder, the stakeholder will observe an internal event of the sub-domain:

**Definition 5** ***(event projection)***: The projection of an event in domain D on sub-domain Z is the event $e_{/Z}$ defined by the projections of states of D before and after the event on Z.

Formally: Let $e=<s_1,s_2>$, $s_1,s_2 \in S(D)$. $e_{/Z} = <s_{1/Z},s_{2/Z}>$.

In the product assembly case, when the domain changes from "order arrived" to "order sent to assembly," the assembly sub-domain changes from "idle" to "busy."

Since several states of a domain may project to a given state of a sub-domain, several domain events might project to the same sub-domain event. For example, all material arrivals might project to a warehouse "storage" event.

Domain transitions, governed by the transition law, may be manifested as events in sub-domains. Thus, the domain law is *projected* on sub-domains:

**Definition 6** ***(law projection)***: The *projection of transition law* $L^D$ of domain D on sub-domain Z (denoted $L^D{}_{/Z}$) is the mapping defined by the events projected on Z when the state of D changes according to $L^D$.[9]

The following example demonstrates a projected law. It shows that even when a domain behaves predictably, its sub-domains do not necessarily behave predictably. Assume that materials arriving from a supplier are stored differently than those arriving from a customer. In either case the projection of the initial state on the warehouse sub-domain (prior to storing) is "materials arrived." If storage actions differ by case, the warehouse will appear to behave unpredictably to someone observing only the warehouse and not the origins of the materials that arrive.

In general, predictable domain behavior does not imply predictable behavior in sub-domains. However, for some domain states, a sub-domain might behave predictably. For example, if a sub-domain represents the actions of an agent (e.g. a business unit, or a product cell) that operates independently under some conditions:

[9] Formally: Let the state of sub-domain Z be u. Let the set of states in D which projects into u be $S^D(u)$. The law on D maps every $s \in S^D(u)$ to $L^D(s)=S'$. The projection of $L^D$ on Z will be the projection of S' on Z, $L^D{}_{/Z}(u) \in S'_{/Z}$.

**Definition 7** ***(predictable behavior)***: A sub-domain Z behaves *predictably* for a subset of domain states S' iff the projection of the transition law $L^D$ on Z maps equivalent sub-domain states (projected from the set S') into equivalent sets.

Formally: For $S' \in S(D)$, each $s \in S'$, $L^D(s)/_{/Z}$ maps into equivalent states in S(Z).

In words, for states in S' $L^D{}_{/Z}$ fully describes the behavior of Z, independently of the values of state variables not in the sub-domain ($X^D$-$X^Z$).

For example, assume that the warehouse always stores and records materials in the same way, independent of the source. The warehouse sub-domain will then behave predictably for all states that trigger storage actions. When a sub-domain Z behaves predictably, the internal transitions in Z depend only on $X^Z$ (and not on the state variables outside the sub-domain ($X^D$-$X^Z$)). Hence, we will also say that the sub-domain Z *behaves independently*. Predictable behavior and independent behavior of a sub-domain have the same meaning.

### *Process Models and Process Paths*

We define an *abstract view of a process*, independent of actual implementation, as state changes in the process domain. In the following, we assume that the process stakeholder's view of the process domain D is represented by a set of state variables – $X^D$. This view is reflected in a lawful state space of the process domain– $S^L$. The actions $\mathcal{A}$ and decisions $\mathcal{D}$ available to the stakeholder are related by:

$\mathcal{D}$: $S^L \rightarrow \mathcal{A}$. The mapping $\mathcal{D}$ induces a *partition* of $S^L$ into equivalence sets,

$\{S_k; k=i\ldots N\}$ such that $\forall s \in S_k$ $\mathcal{D}(s)$ is the same. The stakeholder is aware of a set of external events E that can affect the process.

We first define an enacted (actual) process in the domain D:

**Definition 8** ***(process)***: An enacted *process* is a sequence of state changes in a given domain beginning with an unstable state and leading to a stable state.

**Corollary**: In an enacted process, unstable states change according to the domain transition law, and stable states (except the last) are changed by external events.

Formally: A process is a sequence of states $<s_1, s_2, s_3, \ldots s_m, s_{m+1 \ldots}>$ such that $s_k$, $s_{k+1}$ for $k \geq 1$ are not equivalent and that either internal or external events exist from $s_k$ to $s_{k+1}$: $s_{k+1}=L(s_k)$ or $\exists$e (external event) such that $s_{k+1}=e \otimes s_k$.

Definition 8 is generic and shows an enacted process as modeled in GPM without using an activity construct. In practice, the word "process" is used for the *abstract notion of a process,* referring to a *process class – a set of possible process occurrences considered similar by a stakeholder.* A process class is described by *a*

*process specification*. We formalize such specification in four elements (all considered "classes"):

**Definition 9** ***(process specification)***: A specification of a process in a given domain is a quadruple:

I: the set of *possible initial states* – a subset of unstable states of the domain.

G: the *goal set* – a subset of the stable states reflecting stakeholders' objectives.

L: the *transition law* that specifies the domain behavior.

E: a set of *relevant external events* that can or need to occur during the process.

A process specification includes two types of state changes: internal transitions, reflecting the domain dynamics (abstracted as a transition law), and effects of the environment, abstracted as external events. The law (L) is an abstraction that, in practice, will usually be specified in terms of pre- and post- conditions for process activities, and the rules determining the choices among routing possibilities.

As an example, consider a business that assembles products according to customer orders. A process specification may include:

I: all states that reflect arriving customer orders.

G: the states where an order has been shipped (which requires that it will be first approved, then assembled, delivery shipment arranged, and the order shipped).

L: the business rules determining: (1) whether an order will be approved or not, and (2) the rules that determine the actions required for assembling the product, the order of these actions, and their assignment to workstations.

E: arrival of orders, arrival of components from suppliers, and machine breakdowns.

To assure that every process in the domain can terminate, we posit:

**Assumption 1 (Stability)**: Every unstable state of the domain can be transformed by a sequence of internal and (possibly) external events into a stable state.

Termination might not necessarily assure that the process completed successfully (namely, reached a state in the goal set). To complete successfully, a sequence of states must lead via internal and (possibly) external events to a state in the goal set.

We also assume that every process must be triggered by interactions with the environment (manifested as events external to the process domain but affecting it):

**Assumption 2**: Before the process begins, the domain is in a stable state.

Example: the assembly line is idle prior to a work order arriving.

According to the abstract definition of a process model (Definition 9), process design can be viewed as defining the domain law – L – such that, for a given set of

external events, at least one possible trajectory of states will connect each initial state to a goal state. This abstract view can be linked to concepts such as activities, actors and resources (Soffer and Wand, 2004) that are typically used in process models. Process implementation can be viewed as choosing the actual actors, actions, and resources that will be involved in enacting the specification of the law.

As explained, a process specification defines a ***class*** of processes where each process is an ***enactment*** (an instance of the class). Recall that, for practical purposes, a process stakeholder may consider different states as equivalent. Therefore, the stakeholder may not distinguish between two process class instances that proceed through different, but equivalent states. A special case is when two sequences of activities are performed concurrently and independently (or "in parallel"). For example, in order processing: checking inventory availability, and checking customer's credit. Such activities may be executed in various orders but these differences might not matter to the stakeholder. We formalize this idea using the notion of a *path*:

**Definition 10** ***(process path)***: Given a process specification, a *process path* $p^X$ is a set of possible enactments, where each proceeds through the same sequence of equivalent sets of states (each reachable by a transition or an external event from a previous equivalent set). The first state of each enactment is in the initial set (of unstable states – I) and the last is in the goal set (of stable states – G).[10]

In this definition X can be the whole process domain (D) or any sub-domain ($Z \subset D$).

We will denote the set of states comprising a path $p^X$ by $\{p^X\}$.

It is important to clarify the differences between the three definitions above. Definition 8 (a process) refers to an actual enacted process (process instance). Definition 9 (a process specification) refers to an abstract process *class* (that can have many enactments). Definition 10 (path) refers to the different possible enactments that appear the same to the process stakeholder.

Finally, the notion of *thread* is important for the discussion that follows. In GPM terms, a thread is a sequence of state changes through equivalence sets of any domain (the whole domain or any sub-domain). In the following, we will discuss

[10] Formally, let X be a domain. Let $PR^X=<I,G,L^X,E>$ be a process specification over this domain ($I,G \subset S^L(X)$; $\forall s \in I$, $s \notin L^X(s)$; $\forall s \in G$, $s \in L^X(s)$). Let $\{S_m; m=1 \ldots N\}$ be a partition of $S^L(X)$ to equivalence sets with respect to the transition law $L^X$. A process *path* in X is a sequence of equivalent sets of states $p^X=<S_1,\ldots S_N>$ $S_k \subseteq S(X)$ such that: $S_1 \subseteq I$, $S_N \subseteq G$, and $\forall k=1 \ldots N-1$: $S_k \not\subset L(S_k) \Rightarrow S_{k+1} \subseteq L(S_k)$ or $S_k \subseteq L(S_k) \Rightarrow \exists e \in E$, $S_{k+1}=e \otimes S_k$ (assuming the event transitions equivalent states in the same way).

domains that are decomposable to independently behaving components, i.e. sub-domains. For these cases, a thread will refer to state changes in the lowest level independent components. A complete thread of the whole domain (possibly manifested as concurrent threads of sub-domains), is a process path (Definition 10).

## Analyzing process routing situations

We now apply the GPM view of processes to the analysis of routing behaviors (typically manifested in process models via split and merge nodes). We seek a complete catalog of non-redundant sets of possibilities that are independent of resource constraints or software-related mechanisms. We provide additional formal analysis regarding the completeness and non-redundancy of the catalog in Appendix 1.

### *Analyzing Split Structures*

We identify the phenomena included under the term "split" based on three split cases in the workflow pattern initiative (Aalst et al. 2003). These cases fall into two categories. First, "a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order" (AND-split). Second, "when based on a decision or workflow control data," "either one of several branches is chosen" (XOR-split) or "a number of branches are chosen" (OR-split) (Aalst et al. 2003, pp.10-13). In the AND case all branches must execute. In the XOR and OR cases at least one must execute. The choice depends on the status of the process at the split point.

These descriptions clearly refer to two different phenomena:

(1) *Concurrency*: several threads may proceed concurrently and in any order.

(2) *Choice*: at least one thread of several possible ones must be chosen.

Concurrency and choice can occur separately or in combination. In concurrency, several threads can execute simultaneously. This may occur independently of the process state at the split. In choice, several threads are available, but the state of the process when the choice is made will determine which ones are executed. Often, two phenomena will be combined, and then choice can cause several threads to execute concurrently. In other words, threads exist that can act concurrently, but not all must be activated. There will be no choice if all threads must become active.

We note that c*oncurrent execution* can refer to two types of phenomena. First, relevant to our analysis, several sub-domains can be active concurrently (independently) in the same process instance. Second, it is possible that several instances of a process or of a sub-process may be executed concurrently (e.g., processing several customer orders concurrently). Since this possibility depends on available resources, we do not include it in our analysis. Following the above, we categorize split phenomena based on *choice* and *concurrency*. A choice can be made only when several paths are available for the domain to undergo changes. For example, when a product can be either purchased or manufactured, the process definition should enable two paths, for purchasing and for manufacturing. Such situations require that a given set of states reached in the process can be partitioned into several subsets, each transforming to a different set of (equivalent) states. The choice of path will depend on the values of some state variables, creating a partition of the states at the split. In summary, a choice split is characterized by the existence of a number of paths that might be taken when the process reaches a given set of states of the domain.

Concurrency is enabled through decomposition of the domain to independently-behaving sub-domains. For example, in customer order processing, preparing the goods and coordinating delivery can occur concurrently in two independently operating units, the warehouse and transportation.

We formalize this observation in the following Lemma:

**Lemma 1:** Two sub-domains can transform concurrently (or one transforms and the other remain stable) if and only if they are independent of each other.

**Proof**: Concurrently operating sub-domains can transform through their sequences of states in any relative order. Considering X, $L^{D}_{/X}$ transforming a state within X does not depend on the other concurrent sub-domains. Conversely, when sub-domains behave independently, each can change depending only on its own state. Thus their sequences of states can transform in any relative order, i.e., concurrently.

We term a domain as *decomposable* if it can be decomposed into sub-domains that have independent and concurrent behaviors.

Based on the above analysis, we define a *split point* in a process model:

**Definition 11 *(split)*:** Let S be a subset of domain states on a process path reachable by the same transition or external event. S is a *split* point iff at least one of the following can happen:

(a) The domain becomes decomposable (into independently behaving sub-domains) for every transition possible from every state $s \in S$; or

(b) The set S can be partitioned into *at least* two subsets such that each leads to a different process path. Formally, at least two states, $s_1, s_2 \in S$ exist such that $L(s_1) \cap L(s_2) = \varnothing$ and $L(s_1)$, $L(s_2)$ are not subsets of a set of equivalent states.

The definition is independent of constructs used in process models and refers to the two different "split" phenomena: first – *concurrency*, second – *choice*. The definition implies that either or both cases can occur.

We aim at identifying a full set of *split types* by considering all possible combinations of the two phenomena. While our general considerations will apply to splits of any dimension, we focus on the binary case to accomplish completeness. Multiple process paths can exist whether the domain is decomposable or not. A non-decomposable domain can proceed through one of several alternative paths. Conversely, for a decomposable domain, when all sub-domains can proceed independently and concurrently, there is one path with no selection to be made.

The two phenomena, alternative paths and decomposition, can be combined where different paths entail different combinations of activated (and independently behaving) sub-domains. All split possibilities can be determined by examining combinations of the two dimensions, multiple paths and decomposability, as described in Table 1.

**Table 1: Domain decomposability and multiple paths**

| | | **Multiple alternative paths (decision)** | |
|---|---|---|---|
| | | **No** | **Yes** |
| **Domain decomposability** | **No** | No split (sequence). | Exactly one path must be selected for the domain to traverse. |
| | **Yes** | All sub-domains must be active (concurrently). | At least one sub-domain must be active. |

We summarize all possibilities for the binary case in Lemma 2 below. Some of these possibilities have not been identified as distinct cases so far. We consider each possibility in Table 1. First, when only one path exists for a non-decomposable domain, no split can occur. Second, if multiple alternative paths exist for a non-decomposable domain, exactly one path can be chosen. For example, a person standing at a crossroads can go either right or left, but not both at once. This is an *exclusive choice*, or a split of "XOR" type, where exactly one path can be selected from several available. Since the domain is not decomposable we term this case a *single domain split*. For a decomposable domain, if no alternative paths exist, all

(independent) sub-domains must become active concurrently. Thus, no choice is involved. This is the "AND," referring to several sub-domains that become active concurrently.[11]

Finally, the combination "multiple paths" and "decomposability" indicates a situation where different paths may involve different active sub-domains. Choosing a path in this case implies specifying which sub-domains to activate. A specific case is when exactly one sub-domain can be active. This is a special case of the exclusive choice among domain paths, where the paths differ according to which sub-domain is activated.

In Figure 1 we depict all possibilities for a domain that can be decomposed into two sub-domains. Three possibilities exist. One sub-domain becomes active, or the other sub-domain becomes active, or both become active. In practice, specifying which of the three occurs reflects business rules.

Depending on possible constraints, we obtain the following possibilities:

(1) No constraint. All paths are possible (the "OR" type). Any combination of sub-domains may be activated, but at least one must be.

(2) A constraint implies a choice of only one path. Since a path where only one sub-domain is active does not imply a split, this possibility refers only to the case where all sub-domains must be active (the "AND" type).

(3) A constraint implies a *choice* between several paths. For two domains (A,B) three possibilities exist for such choices:

    a. {A or B}; b. {A or A+B}; c. {B or A+B}

In possibility (3) above, case (a) is a "XOR" split, reflecting domain decomposability where the choice between two paths of activation is a choice between two sub-domains ("two activity sequences"). Cases (b) and (c) represent a choice between activating a *specific* sub-domain and activating both sub-domains. This case can be generalized to a decomposition to N sub-domains $\underline{D}=\{D_k| k=1,\ldots,N\}$ when a given subset $D^*\subset\underline{D}$ of must always be activated. We define this possibility formally.

**Definition 12 *(COR)*:** Let S be a set of states where the domain becomes decomposable to two sub-domains (A, B). S is termed a *Constrained OR* (denoted

[11] In process models, "AND" indicates parallel activity sequences, which mean many execution possibilities for interleaving activity sequences. Our point is that decomposability implies that the sequences are independent, and the outcome would be the same for the stakeholder, independent of the relative order of execution.

"*COR*") split point if for every $s \in S$ a specific sub-domain (A or B) becomes active, and for some states $s \in S' \subset S$ the other sub-domain also becomes active.

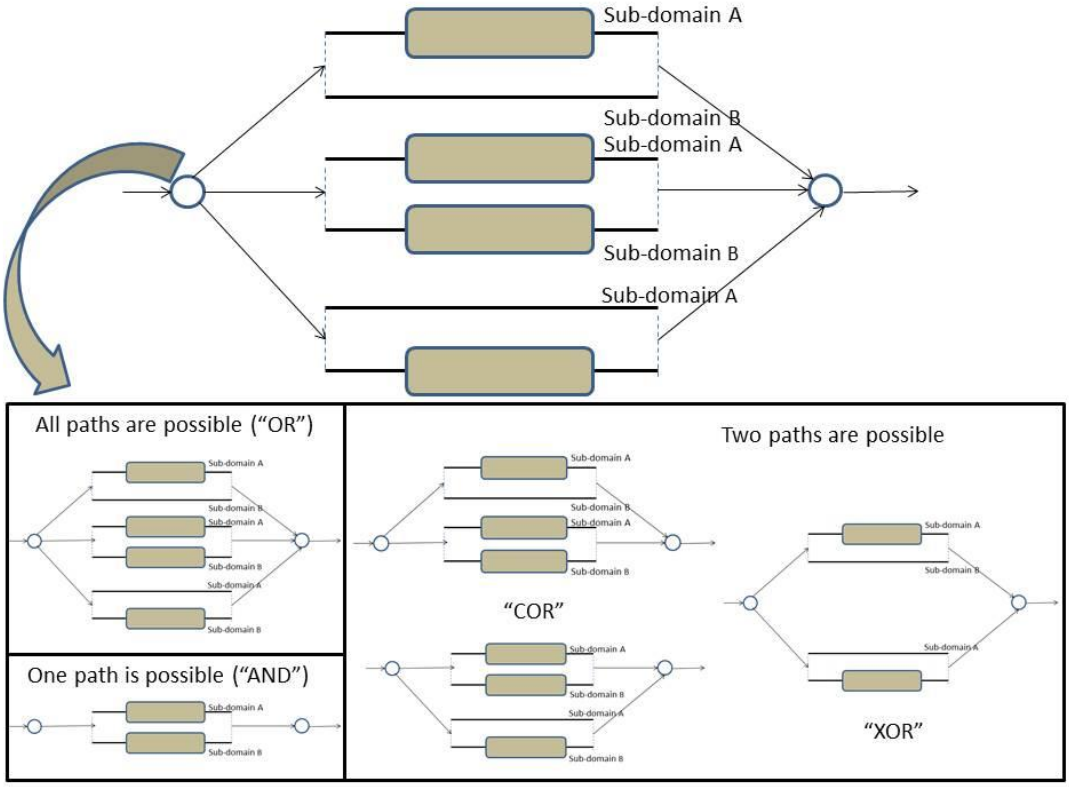



**Figure 1. Possible Paths in a Decomposable Domain (upper part) and their Possible Combinations (lower part)**
**(**shaded rectangles indicate active sub-domains; the circle reflects a choice among alternatives)

We term the sub-domain that is always activated "Mandatory," and the other "Optional." The mandatory sub-domains (e.g. A) can be specified as COR(A). To the best of our knowledge the COR split type has not been recognized as a distinct type. It is possible to specify the COR behavior with logical operators available in process modeling languages (e.g., a combination of AND and XOR, or through conditional flows available in BPMN), and using general specifications such as Causal Nets (Aalst et al, 2011). However, COR is not included as a distinct construct in extant process modeling languages, or in the Workflow Patterns list (Russell et al. 2006). Yet, the behavior described by COR is quite common in practical situations where one action (or several) must always be taken, while other actions might or might not be taken (concurrently with the others). For example,

consider customer order processing where, for most orders, both item availability and customer credit worthiness will be checked, but for some ("preferred") customers, credit need not be checked. Checking item availability is mandatory, while credit checking is optional.

Table 2 summarizes the decomposability-related split types for the binary case.

**Table 2: A Catalog of Decomposability-related split types**

| Constraint | Active domain Possibilities | | | Split type |
|---|---|---|---|---|
| | A | B | A+B | |
| **No constraint** | + | + | + | OR |
| **One path possible** | + | | | No split |
| | | + | | No split |
| | | | + | AND |
| **Two paths possible** | + | + | | XOR |
| | + | | + | COR(A) |
| | | + | + | COR(B) |

A "+" indicates the possible activations at the split point.

Our analysis scanned all possible combinations of the two dimensions in the common definition of split (multiple paths and decomposability) for a binary split. Hence, it assures completeness for binary split types:

**Lemma 2:** Let S be a binary split point. If the domain is not decomposable in S then only a single domain split is possible (the process may follow alternative paths, each potentially involving the whole domain). If the process domain becomes decomposable into two sub-domains in S, then the possible split types are OR (any combination), AND (both), XOR (one exactly), and COR (one is always activated).

Finally, we point out extensions to split of any order:

**Definition 12a *(multi-dimension COR)***: Let S be a set of states where the domain becomes decomposable to N sub-domains: $\underline{D}=\{D_k | k=1,\ldots,n\}$. Let $\mathcal{P}(\underline{D})$ be the power set of $\underline{D}$ and $D^* \subset \mathcal{P}(\underline{D})$ a collection of sets. S is termed a *Constrained OR* split point if for every $s \in S$ all sub-domains in at least one element of D* become active, and for some states $s \in S' \subset S$ additional sub-domains become active.

In this definition, the mandatory sub-domain is replaced by a mandatory choice of a subset of sub-domains. Thus, while an "OR" allows any possible combination of sub-domains to become active, COR specifies constraints on the allowed combinations of active sub-domains.

We identify several combinations of interest of the general definition of COR:

(1) The collection D* contains only one set of sub-domains, which is $D^A$:

1.1 All sub-domains in $D^A$ must be always activated.

1.2 Exactly one sub-domain in $D^A$ must be activated.

(2) D* contains only single-element subsets. At least one of several sub-domains must be activated.

(3) $D^*=\{D^R\}$ and $D^R=\{A\}$. This is a Constrained OR with one necessary sub-domain that can be denoted as COR (A). The binary split is a special case where N=2.

## *Analyzing "merge" structures*

### Defining a merge

A merge implies that a split occurred earlier in the process (or immediately prior to the initial states of the process)[12]. As shown above, at the split, the process domain may be non-decomposable (alternative paths exist for the entire domain) or decomposable (to independently behaving sub-domains). For a non-decomposable domain in which several paths exist, only one path can be enacted at a time. In this case a merge means that the different paths reach the same state (or a set of states that meets some predefined conditions). Thus, merge for a non-decomposable domain can be formalized in terms of sets of domain states:

**Definition 13 *(single-domain merge)*:** Let $\{S_k, k=1\ldots N\}$, N>1, be non-empty sets of states of domain D, such that $S_i \cap S_j=\varnothing\ \forall i,j=1\ldots N, i\neq j$. M is a *single-domain merge* if and only if every $S_k$ includes a state that is mapped by the law into the same set of states in M.[13]

Formally: $\forall k, k=1,\ldots,N, \exists s_k \in S_k , L(s_1)=L(s_2)=\ldots= L(s_N)\subseteq M$.

For a decomposable domain, a variety of merge possibilities can arise. For simplicity and clarity we address only binary decomposition. We will indicate at the end of the analysis how it can be extended in principle to multi-domain cases.

After a split has occurred, the domain is traversing a set of states where each sub-domain can operate independently. We first define a set of states for which independently behaving sub-domains exist:

[12] In many process modeling notations, merge points might be related to loop structures, whereas a continuation might include a repeating set of activities. This has the visual appearance of a merge node located before the split node. Consider how the process actually happens (the sequence of states followed); the merge occurs only after the split.

[13] Every two subsets of states in the definition are disjoint. Hence, at least one subset must include an unstable state. This links a single-domain merge to domain dynamics.

**Definition 14** ***(decomposition set)*****:** Let $\underline{D}=\{D_k,\ k=1\ldots N\}$ be sub-domains of D. $S_{dec}\subseteq S(D)$ is a *decomposition set of states* (with respect to $\underline{D}$), iff $\forall s\in S_{dec}$ each sub-domain $D_k$ (k=1…N) behaves independently of the other sub-domains.
Formally, let $L_{/Dk}(s)$ be the projection of the domain law L on sub-domain $D_k$ (Definition 4). For each sub-domain $D_k$, k=1…N, $L_{/Dk}$ defines predictable behavior (Definition 2) for all states of $D_k$ that are projections of states $s\in S_{dec}$.

For example, assume that after a product is manufactured, two sub-domains A and B operate independently. In A the product is moved into finished goods inventory. In B, a shipment to the customer is arranged. The domain behavior projects changes in the warehouse, and defines the transition law of the inventory sub-domain that is independent of the state of the shipment sub-domain.

Note: in Definition 14 we do not require $D_i\cap D_j=\varnothing$ for $i\neq j$. Two sub-domains can share state variables, but changes in one do not necessarily affect the other.

When a split to two or more independent sub-domains occurs, each sub-domain traverses a path independent of the other. We propose that the meaning of a merge in a process model is a *domain state where at least one of the sub-domains cannot further transform independently*. To formalize this notion, we consider the final states of each of the independent paths traversed by the sub-domains. All sub-domain states for each path, except the last state, are projections of domain states that are in a decomposition set. The last states cannot be in the previous decomposition set, because at least one of the sub-domains stops transforming independently of the other. The decomposition set is valid now only with respect to the sub-domains that remained independent. A merge, therefore, is a set of domain states such that each maps into a state on at least one of the sub-domains in which the sub-domain ceases to be independent.

For example, assume that fulfilling an order involves two independent types of operations, taking place in two independent sub-domains. Order preparation includes assembling, packaging, and preparing goods for loading. Transportation arrangement includes obtaining a truck. The merge comprises all states where the order can be loaded on the truck, namely, “order assembled and truck is ready and awaiting loading.” Once the two sub-domains have completed their tasks, the delivery sub-domain (in which the order is loaded, transported and delivered) becomes active.

We formally define merge following a multiple domain split:

**Definition 15 *(decomposition-related merge)*:** Let $p^{Dk}$ be paths of sub-domains $D_k$, k=1,…,N where at least in the first state of $p^{Dk}$ $D_k$ behaves independently. A decomposition-related *merge* is a set of domain states (M) where at least one of the sub-domains reaches a state in its path where it is no longer independent (other sub-domains might still traverse an independent path).[14]

According to the definition, a decomposition-related merge set (M) can be specified when at least one of the concurrent sub-domain paths reaches states that do not transform independently. In practice, M has some meaning for the stakeholder (e.g. "order can be shipped" or "production can start").

In the following analysis we refer to decomposition-related merges simply as "merge." The merge definition does not prescribe what happens after at least one of the sub-domains reaches M. For the process to continue, the domain should be unstable for some states in M; at least one new sub-domain will become unstable. In the example above, this will be the shipping and delivery sub-domain.

Consider another sub-domain C, different than $D_k$, k=1,…,N, the previously active independent sub-domains. For C to become unstable when at least one of the sub-domains $D_k$ reaches the merge, it must share state variables with $D_k$ as otherwise it will not be affected. However, C should not be part of an independent sub-domain that is still active when others reach the merge.

**Definition 16 *(continuation sub-domain)*:** Let M be a decomposition-related merge of sub-domains $D_k$, k=1,…,N. A *continuation sub-domain* is a sub-domain C for which: (1) $C \not\subset D_k$, k=1,…,N, and (2) C is unstable for at least one state in M.

Formally: $X^C-(X^C \cap X^k)\neq\varnothing$, M is a merge of $\{D_k\}$ and $\exists s \in M$ such that $s_{/C}$ is unstable.

**Corollary**: to assure that the process can continue we require that C shares state variables with at least one of $D_k$, k=1,...,N. Formally: $\exists k \in 1,...,N$, $X^C \cap X^{Dk} \neq \varnothing$.

In the shipping example, M comprises states where the assembled order can be loaded on the truck. The domain C refers to loading, transportation and delivery, and is activated when the order can be loaded.

[14] Formally: Let $p^{Dk}= s^k_1 \ldots s^k_{Mk}$, k=1,…,N , such that for $s^k_1$ $D_k$ has predictable behavior. A decomposition-related merge is a set of domain states, M, where $\exists k \in \{1,\ldots N\}$ $\exists s \in M$ such that: $s_{/Dk} \in p^{Dk}$, and $L_{/Dk}(s_{/Dk})$ is not in a decomposition set.

**Identifying Merge Cases**

Using the above formalization we now analyze possible types of behavior. To simplify the discussion and to accomplish completeness, we focus on a binary merge. We will show that, for this case, the analysis provides a full set of behavior types. Some are recognized Workflow Patterns (Russell et al., 2006) while others have not been previously defined.

We assume that what matters to a stakeholder are the points in time when an organizational actor (such as a person, unit, system or machine) begins taking an (independent) action, or completes an action. We represent such actors as sub-domains. Our analysis of merge will classify each process behavior type in terms of stability and instability of the sub-domains that have become active at the split (A, B), and of the continuation sub-domain (C) that may become active at the merge.

To illustrate, consider an example. Two teams (A and B) are independently engaged in a product design process. The next step in the process will be executed by a third team (C). We illustrate different merge types by the following scenarios.

Scenario 1: When the first team completes the task, C can begin the next development. The work of the other team becomes redundant so it is stopped.

Scenario 2: similar to scenario 1, but the second team is allowed to complete its work and generate an alternative solution that will not be used in the continuation of the process (but might be used in the future).

Scenario 3: regardless of which team completes first, C will start only when both teams complete their tasks, to enable selection of the best solution. This is termed *synchronization,* waiting for two independent threads of activities to end.

Scenario 4: Team A includes experienced experts, while team B is being trained. If team A completes first, its solution will be immediately used for the next task. If team B completes first, they will wait until team A completes so the solutions can be compared before the process continues. This behavior is termed an *asymmetric synchronization* (Soffer et al., 2007) as the need to wait depends on which team completes its task first.

Scenario 5: the two teams perform complementary tasks. The solution of the first to complete is immediately given to the other so they can use it. In this case, the second team does not continue its independent path after the first team reaches its objective. Rather, it takes a different path based on the first team's results. This path can be considered to occur in the continuation sub-domain (C), since it is not independent as before.

The five scenarios demonstrate possible merge behaviors. Each behavior can be specified in terms of the sub-domain that completes its task first, and what happens then to the second and to the continuation sub-domains. Using this specification, we can enumerate all possibilities of merge behavior in terms of two possible events. The first event occurs when at least one sub-domain reaches the merge, namely, ceases to transform independently. It either stops or is no longer independent, and is then part of a continuation sub-domain. A second event will occur if the first event has not stopped the second sub-domain or caused it to change its course (and hence become part of the continuation sub-domain). The analysis allows also for asymmetric cases with respect to the sub-domains. It is possible that when sub-domain A reaches the merge first, what happens to the continuation sub-domain (C) or to sub-domain B is different than what happens to C and A, if B reaches the merge first.

To illustrate, in Scenarios 1 and 2 of the product design example, the process continues when either team provides a solution. The other team is stopped or allowed to complete its task, but the outcome is not used. Thus, only the first event counts. In contrast, in Scenario 3, when one team completes the design (first event), the process continues only when the second event happens. Note that these three types of scenario are symmetric with respect to whether A or B completes first.

Using the two events, we now characterize the domain behavior:

(1) For the first event, by specifying:

a. Whether the continuation sub-domain is activated (becomes unstable) or not; and

b. Whether the other sub-domain proceeds independently or is stopped. If it proceeds, but not independently, it becomes part of the continuation sub-domain.

(2) For the second event, if it is relevant, by specifying whether the continuation sub-domain is activated or not. If not, the process may completely stop.

The second event will occur if and only if:

(1) The continuation sub-domain has not been activated on the first event, and

(2) The other sub-domain was active upon the first event and was “allowed” to proceed independently.

In the product design example, when one team reaches a solution, the next step can either begin or not. The second team may proceed independently, stop, or change their approach. If the second team proceeds but not independently, this is a new (part of the continuing) development phase. If the second team proceeds independently, then a second event will happen. This second event will be relevant

to the continuation of the development process only if the process has not progressed into the next phase when the first team completed its assignment.

We identify all possible merge behaviors as combinations of domain states after the first and second events. Examples are provided in Table 3, where the possible options in the first event are: domain A arrives at the merge, domain B arrives at the merge, or both arrive together at the merge. The second event, when relevant, can only be the arrival of the other domain at the merge. The state after the first event is defined by the states of the continuing domain and of the domain that was still progressing when the event occurred. The state after the second event is defined by whether the continuation sub-domain proceeds or not. Design decisions about these states determine the behavior at the merge point. We designate these decisions by indicating whether the continuing domain remains stable (S) or is activated and becomes unstable (U), and whether the other domain proceeds independently (P) or is stopped (S). When the other sub-domain proceeds but not independently, this is considered part of the activation of the continuation domain.

**Table 3: Some merge combinations**

| | **First event: domain arrives at merge** | | | | | **Second event: arrival of** | |
|---|---|---|---|---|---|---|---|
| | **A** | | **B** | | **Both together** | **A** | **B** |
| **State of domain** / **Case** | **B** | **C** | **A** | **C** | **C** | **C** | **C** |
| 1 | P | U | P | U | U | | |
| 2 | P | U | P | U | S | | |
| 3 | P | U | P | S | U | U | |
| 4 | P | U | P | S | U | S | |
| 11 | P | S | P | U | U | | U |
| 12 | P | S | P | U | U | | S |
| 19 | P | S | P | S | S | U | U |
| 20 | P | S | P | S | S | S | U |
| 38 | S | S | S | S | S | | |
| 39 | S | S | S | U | U | | |
| 40 | S | S | S | U | S | | |

U: unstable; S: stable; P: proceed
Case numbers refer to Table A1 (Appendix 1).

The analysis involves identifying all possible cases and combining all similar cases. Hence, it generates a complete and non-redundant set of behaviors. The full set of cases and a proof of completeness are included in Appendix 1.

As noted, the behaviors are not necessarily symmetric for the two sub-domains. For example, in case 3 in Table 3 (shaded) if A arrives first at the merge, B continues (P) and C is activated (U). If both sub-domains arrive at the merge

together, C is activated (U). If B arrives first at the merge, A continues independently (P) and C is not activated (S). In the latter case a second event occurs when A arrives at the merge. In this pattern of behavior, the difference in outcomes is dependent on which sub-domain arrives first. Further, C is always activated when A arrives (whether first or second), but not when B arrives first. We call this case *asymmetric synchronization where A dominates* (it corresponds to scenario 4 in the product design example above). Case 11 is similar, but the roles of A and B are reversed.

The full list of merge behaviors (Table A1 of Appendix 1) can be simplified in two ways. First, some behaviors are symmetric with respect to sub-domains A and B (e.g., case 3 and case 11 in Table 3). Second, we consider only cases where process completion is assured (no situation can arise that will stop the process). In some cases it is possible that the continuation sub-domain will never be activated (e.g., case 38 in Table 3). In other cases, for every possible split enactment, the process is or might be stopped. For example, in case 19 in Table 3, the process cannot be guaranteed to continue for any enactment. If only one sub-domain is activated at the split, the process will not continue. If both are activated and each one arrives at the merge separately, the process continues on synchronization, but if they arrive at the merge together the process is stopped.

The detailed analysis (Appendix 1) leads to eight generic merge behaviors (Table 4). To demonstrate that all these cases are plausible in practice (namely, have business meaning), we provide examples for each case in Table 4. The "Applicability" column indicates the split enactments for which the merge case can assure process continuation. For example, synchronization is possible only when both sub-domains become active at the split preceding the merge. Similarly, Immediate Continuation with Mutual Blocking is applicable only when one sub-domain is activated at the split. If otherwise, the process might be blocked if both sub-domains become active and reach the merge simultaneously.

The final catalog of eight generic merge types is listed in Table 4 and can be grouped into three categories, each with a different business meaning.

Group 1 (1-3): the process continues unconditionally (on the first merge event).

Group 2 (4-6): specific conditions exist for continuation, indicating that a process has reached the merge through two parallel branches that need to be synchronized. Such synchronization reflects some business requirements.

Group 3 (7-8): situations where certain actions cannot be taken (e.g. blocking may reflect limited capacity at the merge). These cases reflect business constraints that might stop the process. Such constraints may be overcome if the enactment is known before the arrival of a branch at the merge, and the merge type can be dynamically adjusted. These cases may require an appropriate information system.

The last two cases lead to an interesting conclusion. The list of cases is complete with respect to split and merge, when they are considered separately. However, dependency on the actual split enactment may require dynamic merge adjustments, combining different behaviors. Since we have not analyzed possible combinations, they do not appear as cases in our list.

**Table 4: A Catalog of Generic Merge Types**

| | Description and applicability | Applicability | Example |
|---|---|---|---|
| | **Group 1** | | |
| **1** | **Immediate continuation**<br>The process continues when the merge is reached. When both domains are active and one reaches the merge, the other proceeds independently. | All enactments | Two engineers concurrently try to solve a problem. When the first succeeds, the next activity – fixing the problem – begins. The other engineer continues to work on a solution. |
| **2** | **Immediate continuation with cancellation**<br>The process continues when the merge is reached. When both domains are active and one reaches the merge, the other is stopped. | All enactments | Two engineers try concurrently to solve a problem. When the first one succeeds, the next activity – fixing the problem – begins. The other engineer stops working on the problem. |
| **3** | **Immediate continuation with asymmetric cancellation**<br>The process continues when the merge is reached. If a particular domain arrives first, the other is stopped. If the other domain arrives first, the original domain proceeds. In other words, if both domains are active, one will always complete but the other will complete only if it arrives first. | All enactments | Production planning depends on either demand forecasts or on actual customer orders. Forecasts can be prepared while customer orders are sought. Planning can start when the forecast is ready, but orders will still be sought. If orders are available before forecast is completed, planning begins, and forecasting is stopped. |
| | **Group 2** | | |
| **4** | **Synchronization**<br>The process can continue when both sub-domains have arrived at the merge. After one sub-domain arrives, continuation awaits completion of the other (that has been continuing). | When the two sub-domains are active | To process a customer order, both inventory and the credit worthiness of customer must be checked. The order will be processed only when both actions have been completed. |

| Table 4: A Catalog of Generic Merge Types | | | |
|---|---|---|---|
| | **Description and applicability** | **Applicability** | **Example** |
| **5** | **Asymmetric synchronization**<br>The process can continue only when a specific ("necessary") sub-domain arrives at the merge. If the other sub-domain arrives first, the necessary sub-domain must be allowed to proceed independently since continuation requires it. If the necessary sub-domain arrives first, the other sub-domain is allowed to proceed. | When the necessary sub-domain is activated (the other may or may not be activated). | Before production can begin, production planning must be completed. Sometimes, production cost estimates must be done in parallel with planning. However, completion of this activity is not necessary for production to begin. |
| **6** | **Asymmetric synchronization with cancellation**<br>The process continues only when a specific ("necessary") sub-domain arrives at the merge. If the other sub-domain arrives first, the necessary sub-domain must be allowed to proceed since continuation requires it. If the necessary sub-domain arrives first, the other sub-domain is stopped. | When the necessary sub-domain is activated (the other may or may not be activated). | Buyers always seek quotations from a preferred supplier and sometimes also from an alternate. In the latter case, if the quotation from the preferred supplier arrives first, the buyer proceeds to order, and cancels the alternate request. If the quotation from the alternate arrives first, the buyer waits for the quotation from the preferred supplier, then decides from whom to order. |
| | **Group 3** | | |
| **7** | **Immediate continuation with mutual blocking**<br>The process can continue when either domain arrives at merge but not when both arrive together. Hence, for two-domain enactments, continuation of the process cannot be assured. | Only for single domain enactments. No decision needed about the other domain. | Two production lines transfer completed products immediately to the packaging work center, which can handle only one product at a time. If products from the two lines arrive together, they may be damaged. |
| **8** | **Single-sided continuation**<br>The process can continue only upon arrival at merge of a specific sub-domain. Otherwise, it cannot continue. | Only when the specific sub-domain has been activated. | Some products require refrigeration. A refrigeration truck can move all products. A regular truck cannot be used for refrigerated items. If there are such items, and only regular trucks are available, the process will not continue. |

Finally we note that, as in the analysis of split types that yielded cases not previously recognized, the analysis of merge also yielded unrecognized cases. These include cases of both asymmetric and mutual blocking.

### Extending the Analysis to N Sub-Domains

We demonstrate briefly how the merge analysis can be extended to any number of sub-domains. Assume that at a split set of states $S_{Sp}$ the domain D can be partitioned into N independently behaving sub-domains $\underline{D}=\{D_k, k=1,\ldots,N\}$ and that a continuation sub-domain C exists that is inactive at the split.

Assume that $K \leq N$ sub-domains became active at the split. Generalizing the

binary case, we consider a stream of possible events, comprising sub-domains "arriving" at the merge. The arrivals (at most K) will continue until (1) all active sub-domains have reached the merge, or (2) all active sub-domains have been stopped, or (3) the continuation sub-domain has been activated. Each arrival may lead to these decisions:

(1) Whether or not to activate the continuation sub-domain C, and

(2) For each active sub-domain, whether or not to stop it (it becomes inactive).

If an active sub-domain $D_k$ reaches the merge and the continuation is not activated, $D_k$ becomes inactive. If an active sub-domain is proceeding, but not independently, this results in activation of the continuation sub-domain. If the continuation sub-domain has not been activated, the sub-domains that remain active determine the future stream of possible arrival events.

Because the sub-domains behave independently at the split, the state of the domain at each event is described so as to indicate which individual sub-domains are still active and which have reached the merge. Denote this state $\Sigma^D=<\sigma_1 \ldots \sigma_N; s_C>$, $\sigma_k$='P' if $D_k$ proceeds independently and $\sigma_k$= 'S' if $D_k$ is inactive (either $D_k$ was not activated at the split or reached the end state of an independent path). The continuation sub-domain may be stable ($s_C$='S') or activated ($s_C$='U').

Let the state vectors before and after the event be $<\sigma^b_1 \ldots \sigma^b_N; s^b_C>$, and $<\sigma^a_1 \ldots \sigma^a_N; s^a_C>$ respectively. The decision for each event is:

(1) For every $D_k$ such that $\sigma^b_k$='P': whether $\sigma^a_j$='P' or $\sigma^a_j$='S'

(2) Whether or not $S^a_C$ is changed from 'S' to 'U'.

Different cases can be now defined by the possible decisions. We mention only three examples to demonstrate possible combinations of interest:

(1) If C becomes active at each arrival, this will be immediate continuation.

(2) If immediate continuation occurs and some sub-domains are stopped, the result is immediate continuation with selective cancellation.

(3) If a certain set of domains all need to reach the merge for the continuation to be activated, the result will be selective synchronization.

**Repeating Behavior ("Loops") and Multiple Instances**

A common process behavior happens when a sequence of activities re-executes until a certain condition is met, indicating that the process can proceed through new acitivites. In the product development example assume that a team is assigned to solve a problem. If the solution is acceptable upon completion, the process

proceeds. Otherwise, the team is instructed to seek a solution again.

We describe a repeating behavior using the idea of a process path, i.e. a sequence of sets of equivalent states. For a behavior to repeat, two sets of states, $S_1$ and $S_2$, should exist where (1) The first entry of the domain to $S_1$ occurs before the first entry to $S_2$; (2) A sequence of transitions exists from $S_1$ to $S_2$, and (3) $S_2$ can transition into at least two paths, one from $S_2$ to $S_1$ and the other to states in sets that were not visited earlier in the path.

Since there are at least two possible paths from $S_2$, and only one occurs in a given enactment, it is a "choice" (XOR) split (Definition 11 case b). As well, $S_1$ is entered the first time before $S_2$ occurs, and can be entered again after the domain passes through $S_2$. This is a single-domain merge according to Definition 13. In graphical process models, $S_1$ appears as a merge point and $S_2$ as a split point.

Two types of case exist when parts of a process (a sub-process) need to be repeated as multiple instances of the sub-process. An example for the first type is the periodic processing of accumulated bank transactions. In this case, the order of processing transactions is critical (e.g. for daily interest calculations), and hence the transactions will be processed in a loop. This can be modeled as described above. An example of the second type is processing an order for several items. In this case, the sub-process instances can occur concurrently. However, the number of instances that can be handled in parallel depends on the resources available (e.g. number of clerks available to process an order), and such considerations are beyond the scope of our analysis. Still, each repeating sub-process can be modeled using our basic constructs. In summary, the basic cases in our catalog are sufficient for modeling repeating sub-processes.

## Evaluation strategy

The purpose of the catalog (presented in Table 2 for splits and Table 4 for merges) is to provide a list of business situations that involve routing decisions. We suggest that the catalog can be used to help analysts conceptualize such situations. As discussed above, we limited the scope of the phenomena we analyzed. Specifically, we did not include variations of process flows that reflect features of workflow systems (e.g. exception handling and interrupts) or which depend on resources

available, coordination mechanisms, and software capabilities (notably, multiple instances).[15]

We evaluate the catalog on (1) completeness and non-redundancy with respect to its defined scope, (2) being meaningful in business terms, and (3) being useful in supporting conceptualization of routing situations.

Our evaluations can be described in terms of the first three levels proposed by Sonnenberg and von Brocke (2012). At the highest level we justify the problem statement; the need to support conceptualization of process behavior. At the next level we support the specification of the artifact; the catalog of behavior types by a formal analysis (including a proof of completeness). At the third level we evaluate the artifact in an "artificial setting" using examples and by experiments. We have not pursued the fourth level; testing in realistic settings. The methods of evaluation we applied are of two types, *ex ante* and *ex post* (Venable et al., 2012).

Our *ex ante* evaluation included:

(1) Showing through "Mathematical and Logical Proof", by construction, that the catalog is complete and non-redundant given its scope (Appendix 1). The identification of behaviors that are not included in the Workflow Patterns (COR splits and asymmetric, blocking, and single-sided continuation merges) indicates the usefulness of the theoretical analysis.

(2) Showing for all non-standard cases that each behavior in the catalog can be exemplified by a simple but plausible business case (see the examples for the COR split, and the merge examples in Table 4).

Our *ex post* evaluation comprised two experiments to test whether or not the artifact is both usable (with short training) and useful in helping subjects understand routing situations in business processes. These studies are described in the next section.

# Empirical studies

## *Objectives of the studies*

Most previous empirical research on process modeling addressed the quality of the final model (Mendling et al., 2006) or the interaction with a modeling tool (Pinggera

[15] Such mechanisms relate to throughput ("can we serve two customers at once?") and resource loading ("can we process two orders without overloading the same workstation?"). However, they still represent the same fundamental routing decisions we analyze.

et al., 2012). In contrast, our experiments focus on *conceptualizing* the domain behavior *before* the actual construction of a process model. We posit that the catalog can support conceptualization by providing potential *integrated* memory objects. As explained in the section "Cognitive Aspects of Process Modeling," such objects can help recognize and classify a situation. We defined our classification similarly to how analysts conceive of domain behavior. We propose that, based on this classification, the analyst can infer additional information about the situation. These can help to identify additional questions that can lead to better understanding (Savelsbergh et al., 1998). We designed two studies to test this idea.

### ***Study 1***

The first study addressed the impact of using the catalog on the quality of domain conceptualization and on the level of understanding gained by an analyst. Comparing the outcomes of using the catalog to those of using no list at all would have led to two issues: the use of a classification scheme (any scheme) for process conceptualization, and the scheme itself. Therefore, we sought a basis for comparison and used a comparable subset of Workflow Patterns.

The specific research question was, "How well will subjects using the catalog perform a task related to understanding process behavior, compared to subjects using a Workflow Patterns list?" We used two measures. The first reflected success in classifying domain situations that involve process routing decisions. The second measure reflected the inferences drawn about the domain situation after it has been identified as an instance of a specific class of domain behavior.

### *Experimental Setting*

A laboratory experiment was conducted with 54 Information Systems students of a course on Enterprise Resource Planning (ERP) systems and business process design. All participants had taken two modeling-related courses: (1) A systems analysis course where students studied and practiced business process modeling using Event-driven Process Chains (EPC) and Petri nets. In that course students engaged in realistic business process modeling projects using EPC. (2) A systems design course that involved substantial use of graphic modeling techniques.

With respect to the subject population, we note that in an experiment on process model understanding (Reijers & Mendling 2011), professionals could not be distinguished from students. Students with a strong theoretical foundation (notably, in Petri nets) performed better than professionals.

The students worked in two laboratory groups using ERP software. One group (the “Catalog” group of 30 students) used the new catalog. The other (“Workflow,” with 24 students) used a list of Workflow Patterns. The allocation into groups in terms of the students’ modeling ability was random. However, we also tested this statistically (see below).

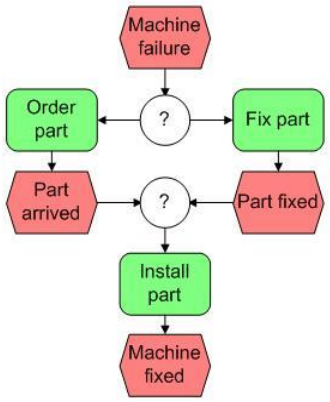
### *Task*

The task comprised two assignments, “Rules” and “Understanding,” for five short situations (example in Figure 2). The Rules assignment had to be done first for each situation (case). Each case included a textual description (Figure 2 (b) and Appendix 3) and an EPC-like diagram, where the logical connectors were left blank (Figure 2 (a)). We used the EPC notation as it was familiar to the participants and therefore there was no need for special experiment-related notation training. We believe the results did not depend on the choice of notation, for two main reasons. First, the task focused on the routing elements that were left blank and hence not affected by the EPC notation (moreover, the behavior was too complicated to be directly expressed with EPC connector types). Second, the purpose was to examine the understanding gained by participants when engaging with the problem, and not to interpret or create a model. Thus, the diagrams served only as *illustrations* to reduce ambiguities that might have existed in the text.

In the first assignment (“Rules”) the students were asked to assign the correct logical rule to each connector in the diagrams, using one of two methods:

(1) Identifying the specific case (depending on the group, either from the catalog or from a list of Workflow Patterns);

(2) Providing a logical expression specifying the behavior of the process at the specific node in a process model fragment (for example, see Figure 2 (d)).




(b) Case description: In a process of handling machine failure, once failure in a machine part is identified, in-house maintenance tries to fix it. If it is very urgent to have the machine operational, a replacement new part may be ordered from the supplier. If the part is fixed before the ordered one arrives, the order is cancelled. If the ordered part arrives before the part is fixed, it is installed, but fixing will be continued and the fixed part may be saved for future needs.

(c) Understanding questions (*answers in italic*):
For each of the following sentences, indicate "true" if it is possible based on the above case description, "false" otherwise, and provide a brief explanation.

1. The machine is working and the maintenance team is still fixing the part
   *True. The ordered part arrived and was installed.*
2. The new part arrived yesterday and the machine is still not working
   *False. The new part should be installed.*
3. The maintenance team is busy fixing the part and the new part arrives
   *True. If the part was ordered.*
4. The order is cancelled and the part is not fixed
   *False. The order should not be cancelled if the part is not fixed.*
5. The part has not been fixed yet and the new part was not ordered.
   *True. If the part was not ordered.*

(d) Split Rule: Always Fix part; If urgent: Order part AND Fix part
Merge Rule: If Fix part only: Simple merge;
If Fix AND Order: If fixed first – Cancelling discriminator, If order arrives first – Structured discriminator

**Figure 2. A situation example (Situation 1) including: (a) Diagram, (b) Case description, (c) Understanding questions (expected answers in italics), (d) Logical rules that can be specified using the Workflow Patterns list.**

The second assignment ("Understanding") referred to the same textual descriptions and included five "true/false" questions relating to the possible process behavior (when enacted). For example, see Figure 2 (c). In this part, the students were also asked to explain their answers.

The Rules assignment preceded the Understanding assignment for two reasons. First, it compelled the students to engage with the models. Second, it served for using the catalog or the Workflow Patterns list as a classification scheme. The Understanding assignment could then reflect students' inferences based on the classification, as an indication of the quality of the mental model they had formed.

***Procedure***

Each group received one hour of training on all cases in the catalog ("Catalog" group) or Workflow Patterns list ("Workflow" group). The training comprised:

(1) An explanation of behavior for each case in the catalog or workflow list. The explanation used the same terms to address the cases in each collection.

(2) Animation (where available) of the process behavior for the cases (based on the Workflow Patterns website (www.workflowpatterns.com)). For the cases in the catalog, behavior was animated when an equivalent workflow pattern was available.

(3) An example was described by the instructor. For the cases that appear in both lists, the examples used for the Workflow and Catalog groups were the same.

(4) A graphic example of the type used in the experimental task was discussed in class. To avoid any effect of differences of training materials (except differences in contents) as provided to the subjects[16], an effort was made to maximize the similarity and appearance of the examples in the Workflow Patterns list to those in the Catalog.

The task was performed immediately after the training session. A printout of the training materials was handed to the participants so they could use it as reference material when performing the task. No time limit was set. To increase participants' motivation, they were promised a quality performance bonus of up to 10 points in the lab component (30%) of the course grade.

### *Task Materials*

The task materials comprised five cases (Appendix 3). Given the purpose of the experiment, we chose cases that enabled comparing our framework to a subset of the Workflow Patterns. This subset represented domain behaviors (split and merge) in process models, but not features dependent on software or implementation. Since the Workflow Patterns collection does not make this distinction, we have analyzed each pattern and identified a subset suitable for this purpose (Appendix 2).

As indicated above, the catalog included cases of split and merge behavior not formally defined previously. Hence, the emphasis in selecting situations for the experimental task was on routing cases that were available in the catalog but not directly in the Workflow Pattern list. These cases could be described by combining patterns from the Workflow Patterns list. However, we wanted to test if any consequences were due only to the added complexity (when a case needs to be combined from other cases), or due to some other issues of the situation described in the task. Therefore, we included two additional test cases. One was directly available in the Workflow list but not in the catalog, and one was directly available in both the Workflow list and in the catalog.

Accordingly, the five situations were:

1. Directly available in the catalog only (situations 1, 2, 5);
2. Directly available in the workflow list only[17] (situation 3);

[16] The full set of training materials is available upon request from the authors.

[17] This case includes a combination of merge behaviors that should be activated based on information about the actual runtime enactment at the split. As explained (see Table 4), the catalog's completeness relates to either split or merge behaviors, each considered separately. However, a choice of cases from the catalog can enable a choice of merge behavior combinations that are based on split enactment information.

3. Available in both (situation 4).

Table 5 summarizes the cases. A situation example is shown in Figure 2, and all the other situations are provided in Appendix 3.

**Table 5: Experimental design**

| Situa-tion | Workflow Patterns (direct match, or logical expression if match not found): Control Group | | The catalog (direct match, or logical expression if match not found): Treatment Group | |
|---|---|---|---|---|
| | *Split* | *Merge* | *Split* | *Merge* |
| **1** | A or (A&B) | When A only: Simple merge.<br>When B arrives first: Structured Discriminator.<br>When A arrives first: Cancelling Discriminator. | COR (mandatory ; A) | Immediate continuation with asymmetric cancellation (if A arrives first, B is cancelled). |
| **2** | AND | If A arrives first: Synchronization<br>If B arrives first: Structured Discriminator. | AND | Asymmetric Synchronization (A should wait for B). |
| **3** | OR | Structured synchronizing merge. | OR | If A and B are active: Synchronization (both teams)<br>If A only or B only: Immediate continuation. |
| **4** | AND | Cancelling discriminator. | AND | Immediate continuation with cancellation. |
| **5** | XOR | If activated branch is known and prepared for. Simple merge.<br>If activated branch is not prepared for. No continuation. | XOR | Single sided continuation (B may not continue if not prepared for). |

The Workflow Patterns referred to in the Table are those listed in Appendix 2:
Simple merge: only one branch is active and the process continues when it is completed.
Structured discriminator: if both branches are active, process continues when the first arrives, and the other branch continues to completion.
Cancelling discriminator: when both branches are active the process continues. When the first arrives, the other branch stops.
Synchronization: when both branches are active, the first to arrive waits for the second to continue.
Structured synchronizing merge: when one is branch active, simple merge occurs. When both branches are active, synchronization occurs.

### *Measurement*

The dependent variables were performance scores on the Rules and on the Understanding assignments. For the Rules assignment, a participant could receive up to 5 points: 2 for correct split specification (1 for partial answers) and 3 for correct merge specification. We assigned the merge specification a higher score because it was more complex and allowed more possibilities for errors. For the Understanding assignment, a student could receive up to 5 points, 1 for each correct answer.

Grading was done by one of the researchers who had not been involved in teaching the course or in the training phase. Since all questions had well-defined answers, marking rules to determine the scores were clear so there was no need for a second coder. Note, if the true/false answer for the Understanding assignment

contradicted the text explanation, the coder relied on the explanation to determine the score (0/1). Examples of textual answers are shown in Figure 2 (c).

### *Controls*

*Assignment of students to groups*: to test whether group assignment could have affected tasks performance, we conducted a one-way analysis of variance (ANOVA) on the average homework grades achieved in the course. For the hypothesis of no difference between the groups, we received a p-value = 0.978 (also Levene's p-value for homogeneity equals 0.547). We therefore concluded that assignment to groups was random with respect to students' ability to perform the tasks.

*Materials*: we took several measures to ensure the materials for both groups were as similar as possible. In particular, the cases in the training materials (that served as catalogs or lists for the task) were described using the same terminology. In addition to the three situations unique to the catalog, a further control was included by choosing situations that were either available in both the workflow list and in the catalog, or only in the workflow list.

*Grading*: As explained above, our grading scheme scored 2 points for a correct split specification and 3 points for a correct merge specification. We believed this reflected the relative difficulty of providing answers (in terms of possible errors). However, to find whether this weighting decision might have affected the results, we repeated the data analysis with equal weights given to split and merge. There was no difference in the conclusions.

### *Analysis and Findings*

All task situations appeared in either the catalog and/or the workflow list and can be partitioned into two groups with respect to the Workflow Patterns list. Situations 1, 2 and 5 appeared in the catalog but not in the Workflow Patterns (but could be constructed as a combination of existing patterns). Situations 3 and 4 appeared in the Workflow patterns List. Situation 3 did not exist in the catalog (but could be combined from catalog entries). Situation 4 existed in both collections.

**Table 6: Performance means, standard deviations, p-values**

| **Situation*** | | **Rules assignment** | | | **Understanding assignment** | | |
|---|---|---|---|---|---|---|---|
| | | **Workflow group** | **Catalog group** | **p-value** | **Workflow group** | **Catalog group** | **p-value** |
| 1 | mean (st.dev.) | 2.000 (1.504) | 4.567 (0.817) | 0.000 | 4.420 (0.930) | 4.800 (0.484) | 0.05 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | mean (st.dev.) | 3.583 (1.586) | 4.867 (0.434) | 0.000 | 4.417 (0.717) | 4.667 (0.479) | 0.112 |
| 5 | mean (st.dev.) | 1.917 (1.412) | 3.900 (1.125) | 0.000 | 4.208 (0.833) | 4.600 (0.563) | 0.041 |
| *1,2,5* | *mean (st.dev.)* | *2.500 (1.121)* | *4.444 (0.505)* | *0.000* | *4.347 (0.586)* | *4.689 (0.289)* | *0.017* |
| 3 | mean (st.dev.) | 4.625 (1.135) | 4.667 (0.802) | 0.555 | 4.583 (0.584) | 4.767 (0.504) | 0.17 |
| 4 | mean (st.dev.) | 4.500 (1.022) | 4.867 (0.571) | 0.121 | 4.667 (0.482) | 4.667 (0.547) | 0.863 |
| *3,4* | *mean (st.dev.)* | *4.563 (0.838)* | *4.767 (0.612)* | *0.435* | *4.625 (0.397)* | *4.717 (0.340)* | *0.416* |

** Situations 1, 2, and 5 appear in the catalog but not in the Workflow List. Situation 3 appears in the Workflow List but not in the catalog; Situation 4 appears in both.*

Table 6 compares the performance measures for all five cases. The table provides the means and standard deviations for each situation and averages for cases that appeared (1, 2, 5) and cases that did not appear (3, 4) in the Workflow list.

To test whether observed differences were statistically significant we used a non-parametric Mann-Whitney test, as the grades were not normally distributed.

For the Rules assignment and the three cases available only in the catalog, the Catalog group performed considerably better than the Workflow group. This applies to each individual case and to the average over the three cases (4.44 of 5 for the Catalog group, 2.5 of 5 for the Workflow group). The one-sided non-parametric Mann-Whitney test indicated high statistical significance (p-values of 0.000).

For the two cases that were available in the Workflow list, the average performance was similar in the two groups (4.56 for the Workflow group and 4.77 for the Catalog group). The differences for each of the two individual cases and for their average were not statistically significant (on a two sided test).

For the Understanding assignment and the three cases available only in the catalog, the Catalog group performed better than the Workflow group on the individual cases and on their average (4.69 of 5 for the Catalog group and 4.35 of 5 for the Workflow group). In a one-sided non-parametric Mann-Whitney test the differences for each case were found statistically significant at the 5% level for two of the three cases (1,5) and at less than 2% for the average over the three cases.

For the two cases that were available in the Workflow list the average performance was similar in the two groups (4.625 for the Workflow group and 4.72 for the Catalog group) and the difference was not statistically significant.

Table 6 indicates that the mean grade achieved in the Understanding task for all situations (in both groups) is higher than 4. This implies that the understanding of domain behavior was good. Still, the Catalog group achieved higher grades. A statistically significant difference was found for two of the cases available only in the catalog (1 and 5), and for the average over the three cases.

Finally, to find whether the ability to classify the behavior rules is indeed related to better answers of the understanding questions, we analyzed the correlation between the Rules and Understanding scores. We found a positive correlation with $R^2$=0.229 (significance: p=0.0003). This correlation can be considered as approximately medium. Yet, given the generally high grades with low variance of the Understanding assignment, it indicates that success in classifying a situation can also imply understanding and inference about the detailed behavior in the situation.

***Study 2***

The first study provided evidence that the catalog had advantages over a Workflow Patterns list, assuming each served to classify routing behavior. This study did not provide evidence about the actual use of the catalog in classifying and conceptualizing behavior. The second study was intended to obtain the actual thinking process. We performed a think-aloud protocol study, in which subjects verbalize their thoughts as they perform a task. The verbalization is then qualitatively analyzed. To understand the impact of the catalog, we compared its use to task performance based on the use of process modeling knowledge. Specifically, we wanted to find if using the catalog as a classification scheme required additional effort in comparison to using the basic building blocks of process modeling languages.

***Experimental Setting***

Participants were seven Information Systems students attending an advanced course on business process management. Such number is considered appropriate for think-aloud studies, since a qualitative understanding is sought rather than statistical significance (Nielsen, 1994). Participants had previously studied and practiced business process modeling using Event-driven Process Chains (EPC), Petri nets, and YAWL, and were introduced to Workflow Patterns. The task was similar to that of the first study. We used the same five cases, and added a non-binary split and merge case (see appendix 3) to test the ability of subjects to infer from the binary catalog cases to more complicated situations.

Three participants (the “Catalog” group) were trained similarly to the "catalog" group of the first study and were asked to use the catalog. Four participants (the “No Catalog” group) did not use any list. They were trained for performing the task using the same examples as the other group, but without presenting these examples as a reference list. The task was performed separately by each participant, with no time limit. The verbalization was recorded and transcribed. To increase participants’ motivation, the grade of the assignments was 5% of the course grade.

***Analysis and Findings***

We have analyzed the transcribed text using open and axial coding (Strauss and Corbin, 1998). The open coding involved breaking the text down to segments and assigning each segment a category reflecting its use in solving the problem. The axial coding involved grouping the categories into higher-level aspects of the solution process. The resulting categories appear in Table 7**Error! Reference source not found.**. We counted the occurrences of segments in each category and averaged the counts (over the cases) for each of the groups. **Error! Reference source not found.** Table 7 shows the results together with the average performance score.

**Table 7: Summary of the findings of Study 2**

| | Rules | | Understanding | |
|---|---|---|---|---|
| | Catalog | No catalog | Catalog | No catalog |
| Performance score | 4.67 | 4.29 | 4.83 | 4.25 |
| Explicit difficulty expression | 0.06 | 0.79 | 0.06 | 0.33 |
| Use of "key words" | 7.33 | 3.42 | 1.67 | 0.54 |
| Evaluating alternatives | 0.56 | 0.29 | 0.06 | 0.00 |
| Revisiting text (per modeler) | 1.67 | 2 | 0.33 | 3.25 |
| Back out of previous answer (per modeler) | 0.33 | 0.50 | 1.00 | 2.00 |

We now explain the categories and discuss the results for each.

*Explicit difficulty expressions*: These are explicit indications of difficulty, using expressions such as "Oh, this is problematic… it is complicated.” The “No Catalog” group expressed more difficulties than the Catalog group. On average the “No Catalog” subjects expressed difficulty 0.79 times per case for the Rules assignment and 0.33 times for the Understanding assignment. In comparison, the Catalog group averages were 0.06 for both assignments. This supports our expectations.

*Use of "key words"*: This is the use of concepts taken from the catalog (the Catalog group) or from process modeling vocabulary (the “No Catalog” group). The use of

"key words" indicates classifying a situation into a known scheme. For example, "…once the sales report is ready we have an immediate continuation with cancellation and then…" For both assignments, the Catalog group used "key words" much more frequently than the No Catalog group. This indicates that the catalog was indeed used (and more than standard concepts) for classifying the given situations.

*Evaluating alternatives*: these are situations where participants systematically considered alternative solutions, evaluated them, and selected one. Alternatives usually related to "key words," and indicated a systematic thinking process guided by the catalog concepts or by standard constructs. For example, "… so we can have immediate continuation… no, we need with cancellation…no, but this should be an asymmetric cancellation…" Alternatives were evaluated mainly in the Rules assignment (0.56 times per case by the Catalog group, 0.29 by the No Catalog group). Notably, we counted the occurrences of alternatives evaluation, not the number of alternatives considered.

*Revisiting text*: In these situations, participants returned to the case description while attempting to answer a question. We interpreted this as indicating a lack of clear understanding of the case, or a lack of a suitable model for it in working memory. Because revisiting text did not occur often, we report in Table 7**Error! Reference source not found.** the average occurrences per modeler (not per case). These numbers were similar in both groups for the Rules assignment (where cases were classified). However, a large difference existed for the Understanding assignment: 0.33 for the Catalog group vs. 3.25 times for the No Catalog group. We believe this outcome reflects the advantage of the catalog as a classification scheme. When a case is classified, it can be more easily stored in working memory without a need for the case details.

*Back out of previous answer*: In these situations, the subject gave an answer, and later realized that the answer needed correction. Due to the low numbers, these situations are reported per modeler. They occurred more frequently in the No Catalog group than in the Catalog group, especially in the Understanding assignment (2 per modeler in the No Catalog group, 1 for the Catalog group).

This study did not focus on performance. However, we also checked the score of correct answers (similar to the first study). The scores of the Catalog group were higher than those of the No Catalog group. Although it is not possible to check the statistical significance of these results, we believe these findings are in line with the

findings of Study 1 that indicate that the catalog supports domain conceptualization. This conclusion links (non-statistically) the qualitative findings to performance.

Finally, considering Case 6 of the non-binary split and merge, we looked for evidence of increased difficulty or for insights how the binary cases of the catalog were used for understanding more general cases. We found no difference in the performance score for this case compared to the binary cases (for both groups, both assignments). However, there were more expressions of difficulty by the No Catalog group than in the Catalog group, and a higher use of "key words" by both groups. This provides an early indication that the catalog concepts, while defined for binary cases, may not be difficult to extend and apply to more complicated situations.

### *Summary of the Empirical Results*

The two studies complemented each other and addressed both the quantitative performance aspect (where the catalog was compared to Workflow Patterns) and the qualitative process aspect (where the catalog was compared to the use of standard modeling constructs). Both studies included the Rules and Understanding tasks (in that order). These tasks were intended to test our suggestion that the catalog can serve as an effective classification scheme when forming a mental model (Derry, 1996; Larkin, 1985). Classification, achieved through the Rules assignment, related a situation to a general case (that can be considered an integrated memory object). The Understanding assignment tested inferences, namely the ability to understand or predict specific details based on an identified class. For example, classifying the split at Situation 1 as COR helped subjects understand (by inference) that a new part might or might not be ordered, but the old one would always be fixed.

Our findings indicate that the catalog can support conceptualization of domain behavior with respect to routing phenomena. Via this classification and inferences, the catalog can lead to better understanding and to recognizing the need for more information about the case.

In Study 1, the catalog performed better for cases that were not directly available in the Workflow Patterns list, and at a comparable level for cases that appear in both (or not directly in the catalog). Perhaps it was not surprising that the classification was better supported by the scheme that includes cases not directly included in the other. This is consistent with the need to minimize the cognitive load caused by integration (Paas et al., 2004). However, the case that was directly

available as a workflow pattern and not in the catalog did not result in better performance for the Workflow group. This might indicate that the concepts in the catalog make classification easy enough to overcome additional integration effort.

While the differences in performance results for the Understanding assignment were statistically significant, the effect appeared rather small in magnitude (about 8% on average). However, one has to be careful in interpreting the practical significance of such results. It can usually be assumed that a modeler has a good understanding of the domain before constructing a model. Hence, it could be expected that both groups would perform well on the Understanding questions. This is likely demonstrated by the relatively high scores. Practically, however, it is important to consider the number of *errors,* rather than only the correct answers. Errors in the analysis might lead to costly outcomes (both in business results and in efforts to correct existing processes and applications). The number of errors of understanding is the difference between the maximal value of 5 and the score obtained (Table 6). Based on this value, the Catalog group made an average of 50% less errors than the Workflow group.

The effectiveness of the catalog was further indicated by examining the standard deviations of performance scores within each group. Those were consistently lower for the Catalog group than for the Workflow group. This indicates a higher convergence of understanding and more consistent analysis in the Catalog group. We have not hypothesized about such differences, and did not test their statistical significance. However, we believe that this further indicates that the catalog allows better performance than the Workflow Patterns.

Finally, the qualitative findings of the second study provide some insights about the impact of using the catalog. These findings support our claim that: (1) The catalog can be used as a classification scheme that supports inference when a detailed understanding is needed; and (2) The catalog entries can help reduce the cognitive effort of domain behavior analysis. The additional case, with a non-binary split, indicates that catalog cases can be readily extended to more complicated situations.

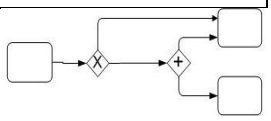
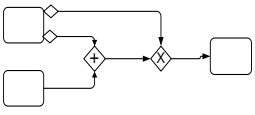
## Discussion

We now present possible uses of the catalog, compare the approach we used for its development to two possible alternatives, discuss limitations of the work, and briefly describe the work in design science terms.

### *Using the Catalog*

The catalog can be used to support process modeling in two ways. First, it can help analysis and conceptualization of routing situations by providing a classification of such situations. Once a situation is classified, the analyst can identify additional questions related to it and explore it further. Second, given specific process modeling constructs, a combination of constructs can be specified for each class of behavior. Thus, the catalog can be used both for exploring and for mapping process behavior. Classification can be done in two ways. First, each class can be defined by intension as criteria to be sought about a situation. Second, a class can be specified by extension, as a list of typical instances. The full details are beyond the scope of this paper. However, we demonstrate this application in Table 8 using examples, of a split case and of a merge case. The table also includes examples for guiding model construction using BPMN notation (Wolf and Soffer, 2014).

**Table 8: Examples demonstrating the application of the catalog**

| **Type of Case (Class of routing behavior)** | **Definition by Criteria** | **Additional information required** | **BPMN Representation** |
|---|---|---|---|
| "Split" | | | |
| COR (M) | (a) two independent sub-domains exist.<br>(b) one sub-domain always activates.<br>(c) cases exist where the other is not activated. | Identify the sub-domain that always activates. |  <br>Note: other representations are also possible. |
| "Merge" | | | |
| Asymmetric synch | (a) two independent sub-domains exist.<br>(b) a continuation sub-domain exists.<br>(c) the continuation sub-domain activates only when a specific sub-domain reaches the merge. | Identify the sub-domain that activates the continuation. |  |

### *Possible Alternatives for Theoretical Development*

We used the GPM view of processes, which represents the *dynamics of business domains* in terms of states, events, and transition laws. This representation

enabled analyzing the dynamics of a business domain, in terms of activating sub-domains and stopping active sub-domains. The analysis enabled us to identify various types of routing behavior, including patterns not formerly defined (manifested in asymmetries, cancellations, and blocking). It would be interesting to consider whether or not alternative representations of process dynamics could have been used for the same purpose. We refer here to Causal Nets and to Petri Nets, as both enable representing aspects of domain dynamics, and to the CASU approach that was used for developing a list of process instantiation possibilities in terms of state conditions and events.

Causal Nets (C-Nets, Aalst et al, 2011) are graphs "where the nodes represent activities and arcs represent causal dependencies" (p. 30). C-Nets can be used to characterize how a given combination of activities starts another activity. This may in turn engage in combinations of activities, leading to other activities. Thus, C-nets can model the conditions governing the flow of activities in a process, and signify the start and stop events of activities. However, for our purpose, C-nets lack two aspects that were important: (1) A full set of possible actions (such as cancellation of activities), and (2) The full definition of a state in terms of *relative times* at which activities may complete. Using relative times enables the definition of synchronization, selective (possibly asymmetric) continuation, or cancellation. Thus, analysis based on C-nets would not provide the full set of behaviors that we identified.

Petri Nets (PNs) and their specialization to Workflow Nets have been used to model the dynamics of business processes (Aalst, 1997) and can in principle be used to describe complex behavior. However, in contrast to GPM and to Causal Nets, the elements of PNs (places and transitions) do not necessarily map to well-defined aspects of business domains. Thus, to ensure that all the structures addressed by a PN-based analysis represent meaningful behaviors, the PN constructs would need to be mapped to domain concepts. This was done in a previous work (Soffer et al. 2010). The mapping requires first a description of the process domain in terms of sub-domains and their state variables. GPM provides such a description directly. Thus, to use PNs for our analysis we would have needed to "transition" via GPM. As well, an important aspect of our analysis of merge included the possibility that an active sub-domain could be stopped. It is unclear how this would be directly represented in a PN.

Finally, we briefly note on the CASU model of process instantiation (Decker and Mendling, 2009). CASU defines various types of initial conditions for a process to be instantiated, and how events that occur at process instantiation are handled during enactment. CASU provides a catalog of possibilities for modeling this instantiation in terms of initial conditions and events. It might be possible to map some of the CASU cases to our merge cases (or *vice versa*), where activating continuation can be comparable to CASU instantiation. However, our catalog is intended to support conceptualization of routing behaviors, while the purpose of CASU is to explore mechanisms in process modeling languages that can reflect the actual instantiation and "use" of events.

***Possible Limitations of the catalog***

The catalog addresses a specific, well-defined scope of domain phenomena. It was shown to be complete with respect to this scope. However, the specific scope has led to possible limitations. First, we did not address aspects that are resource or implementation dependent. We claim, however, that for our purpose of supporting conceptualization, this does not limit the applicability of the catalog. Second, while most of the definitions in the paper are applicable to decomposition to N sub-domains, we limited the detailed catalog to binary splits and merges, where the domain can be decomposed into two independent sub-domains at most. We claim that for several reasons, the results of the analysis are still useful in two main ways.

First, we have shown how the analysis can be extended to more complicated situations by both combining basic behaviors in the catalog and by extending to higher order cases. An example for combining cases is the structured synchronizing merge available in the Workflow Patterns collection (if both sub-domains are active synchronization is required, while if only one sub-domain is active the merge will be of immediate continuation). This behavior was included as Situation 3 in the studies, and in Study 1 exhibited no significant performance differences between the two groups. Our Study 2 provided evidence that extensions to higher order can be done by analysts while conceptualizing a business situation.

Second, even the binary analysis led to identification of cases not included in the Workflow Patterns collection (e.g., COR for binary split, the asymmetric and blocking cases for binary merge).

We note that some cases that appear directly in the Workflow Patterns can in principle be addressed by our analysis (e.g., repeating behavior), but were not

tested in our empirical studies. Their conceptualization can be the subject of further empirical studies.

In summary, both practice and theoretical considerations show that binary cases are useful and provide a basis for more complex cases. The limited but well-defined scope enabled us to prove the completeness of the catalog. From a theoretical point of view, this is an important result that, to the best of our knowledge, has not been previously achieved.

***A Design Science Perspective***

From a Design Science perspective, the catalog and its use can be considered as a *method* artifact (March and Smith, 1995). We suggested that difficulties associated with mapping routing behaviors arise from difficulties in conceptualizing domain behavior when a process may take various paths or threads, or when paths or threads merge. Accordingly, we proposed that the difficulties may be alleviated using a classification of the phenomena in terms understandable to an analyst. Based on cognitive theories we predicted that such a classification can help the analyst identify, conceptualize, and understand a situation. Figure 3 depicts this idea as a design science theory.

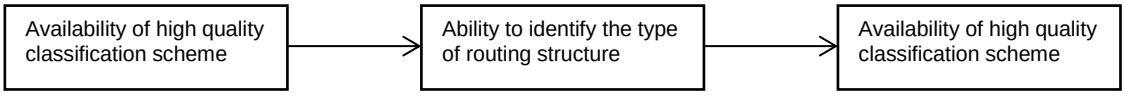



**Figure 3: Predicting the impact of a high quality classification scheme**

We map our work as a design theory using the components proposed by Gregor and Jones (2007) in Table 9.

Table 9: Describing the Catalog in Terms of Design Science Theory

| Design theory component | Catalog development mapping |
|---|---|
| 1. Purpose and scope | Develop a classification of main situations modeled as routing elements in business process models. |
| 2. Constructs | Domain, sub-domain, state, event, law, process thread, path, split, merge, etc. |
| 3. Principles of form and function | A classification scheme of split and merge behaviors is provided to enable an analyst to:<br>1. Ask questions to identify the situation as an instance of a class,<br>2. Ask more questions based on the identified class to fully understand the situation, and<br>3. If a modeling grammar is given, identify the pattern of grammar constructs for mapping the situation to. |

| | |
|---|---|
| 4. Artifact mutability | The original artifact is a set of descriptions of case types in terms of domain behavior. The artifact can be mapped into patterns in different modeling grammars. The artifact can be embedded in modeling support tools. |
| 5. Testable propositions | 1. With the catalog, a business analyst can better identify and understand a given domain behavior than otherwise.<br>2. All cases described a “pure” domain behavior in split and join nodes of process models can be classified as one of the cases in the catalog. |
| 6. Justificatory knowledge | The role of memory objects; cognitive aspects of classification; ontological concepts of domains. |

## Conclusion

Process modeling is important for analyzing, designing and improving business processes, and for developing information systems. Quite a few process modeling languages have emerged and a considerable effort has been devoted to formal analysis of process behavior. However, both practice and research have demonstrated that analysts face difficulties in constructing business process models in situations where decisions need to be made about routing structures, often manifested as nodes of splits and merges.

In this work we propose that major sources of the difficulties are the abstract nature of process routing, and a lack of an appropriate set of concepts for conceptualizing these abstract phenomena. These lead in turn to difficulties forming an accurate mental model of these situations. To facilitate conceptualization, we proposed using a catalog of generic behaviors described in terms readily understandable to analysts. We developed such a catalog, proved its completeness theoretically for the binary case, demonstrated its entries by practical examples, and tested its use in two experimental studies. The studies provided evidence that the catalog is usable and can support understanding of process behavior by modelers.

The main practical use we propose for the catalog is to identify and classify routing decisions in business processes. We believe that the findings that indicate a better process understanding due to the ability to classify situations are important and non-trivial. The questions in the Understanding assignment reflected the domain understanding that should be achieved before constructing a process model. Clearly, to construct a model that completely and accurately represents domain behavior a modeler must understand this behavior. Our findings indicate that this

understanding cannot be taken for granted, and that a classification framework like the catalog can support the required understanding.

The contributions of the work are to theory, to methodology, and to practice. From a theoretical point of view, the analysis provides both a method for identifying and a proof of completeness for routing phenomena (in the binary case) when implementation considerations, resource constraints, and software features are not included. The analysis was done by considering process behavior in terms of state transitions rather than activities that are usually the main construct of process modeling languages. The use of activities in process models may lead to two concerns. First, there is an issue of "granularity" in modeling. It is not always clear where the "boundaries" of an activity lie. Should it be modelled as one activity, or more? Second, while process models are often intended to provide abstractions, activity definitions often reflect how an activity is actually performed rather than what it is intended to accomplish. The abstract view of a process in terms of state changes provides precise definitions, independent of implementation, but anchored in an operationalization of a stakeholder's view. Moreover, this view enables integrating goals and effects of the environment into the abstract model.

From a methodological point of view, the experimental studies show how process conceptualization can be studied without engaging in actual modeling. The link to process models might confound understanding with language-specific considerations.

From a practical point of view, the catalog can help improve education and practice of process modeling. As well, the new cases discovered (both for split and for merge) point to additional behavior patterns that can be supported by modeling languages and process-aware information systems.

We briefly note on the differences between our catalog and the Workflow Patterns collection (Aalst et al., 2003; Russell et al., 2006). The Workflow Patterns were identified in a "bottom-up" approach by "comprehensive evaluation of workflow systems and process modeling formalisms" (Russell et al. 2006). While pragmatic, practice-oriented, and useful, this approach cannot assure completeness. It might lead to redundancy, and to inclusion of patterns that reflect software features rather than process structure. Thus, using Workflow Patterns to understand domain behavior might both confuse an analyst and confound the early stages of analysis with implementation considerations. In comparison, the catalog was constructed

using theoretical considerations, and led to a complete and non-redundant classification with respect to a well-defined but narrower scope,

As discussed above, the work has several limitations that point at several research directions. On a theoretical level, a more complete analysis can be done for cases higher than binary. Such analysis might include more complicated behavior patterns. As well, rules to guide the choice of combinations of split and merge points would be particularly interesting.

From an empirical point of view, it would be interesting to test how the catalog could support the complete modeling process, including the actual mapping into modeling constructs. A first attempt used BPMN routing patterns based on the catalog and showed positive performance (Wolf and Soffer, 2014). It would also be interesting to examine such outcomes in realistic case studies.

At the practice level, future research could map the catalog to constructs available in extant modeling languages, as well as make templates for applying it. Some work in this direction has already developed a semantic interpretation of Petri nets based on the notions of state changes of domains (Soffer et al. 2010).

***Acknowledgement**: Yair Wand is grateful to the Natural Sciences and Engineering Research Council of Canada for supporting this research.*

## References

Aalst van der, W.M.P. (1997). Verification of Workflow Nets. in *Application and Theory of Petri Nets*, P. Azéma and G. Balbo (eds.), LCNS 1248, Berlin: Springer-Verlag, pp. 407-426.

Aalst van der, W.M.P. (1999). Formalization and Verification of Event-Driven Process Chains, *Information and Software Technology* 41(10), pp. 639-650.

Aalst, van der, W.M.P., Adriansyah A., van Dongen B. (2011). Causal Nets: A Modeling Language Tailored towards Process Discovery, Concurrency Theory 2011, LNCS 6901, Berlin: Springer-Verlag, pp. 28-42

Aalst van der, W.M.P., Hofstede, A. H. M., Kiepuszewski, B., and Barros, A.P. (2003). Workflow Patterns, *Distributed and Parallel Databases* 14(1), pp. 5-51.

Aalst van der, W.M.P. & ter Hofstede, A.H.M., (2005). YAWL: Yet Another Workflow Language, *Information Systems* 30(4), pp. 245-275.

Baskerville, Richard L. & Pries-Heje, Jan (2010). "Explanatory Design Theory," Business & Information Systems Engineering: Vol. 2: (5), pp. 271-282.

Bunge, M. (1977). *Treatise on Basic Philosophy: Vol. 3, Ontology I: The Furniture of the World*, Boston: Reidel.

Bunge, M. (1979). *Treatise on Basic Philosophy: Vol. 4, Ontology II: A World of Systems*, Boston: Reidel.

Chandler, P., & Sweller, J. (1991). Cognitive load theory and the format of instruction. *Cognition and Instruction,* 8, pp. 293-332.

Decker, G. & Mendling J., (2009). Process instantiation, Data & Knowledge Engineering 68, pp. 777–792.

Derry S.D. (1996). Cognitive Schema Theory in the Constructivist Debate, *Educational Psychologist* 31(3/4), pp. 163-174

Dijkman R. M., Dumas M., and Ouyang C. (2008). Semantics and Analysis of Business Process Models in BPMN, *Information and Software Technology*, 50(12), pp. 1281-1294.

Figl, K., Mendling J., Strembeck M., Recker, J. (2010). On the Cognitive Effectiveness of Routing Symbols in Process Modeling Languages. *Business Information Systems (BIS)*, Berlin, Lecture Notes in Business Information Processing.

Gregor S. & Jones D. (2007). The anatomy of a design theory. *Journal of the Association for Information Systems* 8(5), pp. 312–335

Hevner, A.R., March, S.T., Park, J., and Ram, S. (2004). Design science in information systems research, *MIS Quarterly* 28 (1), pp. 75-105.

Jonassen, D.H. (2000). Computers as Mindtools in schools: Engaging critical thinking. Columbus, OH: Merrill/Prentice-Hall.

Kindler, E. (2006). "On the Semantics of EPCs: Resolving the Vicious Circle", *Data and Knowledge Engineering* (56), pp. 23-40.

Kuechler Jr., W. L. & Vaishnavi, V. K. (2012). Characterizing Design Science Theories by Level of Constraint on Design Decisions. *DESRIST 2012*, pp. 345-353.

Larkin, J.H. (1985). Understanding, problem representation, and skill in physics. In S.F. Chipman, J.W. Segal, & R. Glaser (Eds.), *Thinking and learning skills (Vol. 2): Research and open questions*. Hillsdale, NJ: Erlbaum. Pp. 141-160

Limonad, L., Varshney, ., L. R., Oppenheim, D. V., Fein, E., Soffer, P., Wand, Y., Chee, Y.M., Gavish, M., Anaby-Tavor, A. (2012). The WaaSaBE model: Marrying WaaS and Business-Entities to Support Cross-Organization Collaboration Using Commitment-Centric Analysis, *Proceedings of SRII Global Conference* ("IT-Enabled Services"), San Jose, California, July 2012. pp 303-312.

March, S. T., and Smith, G. F. (1995). Design and Natural Science Research on Information Technology, *Decision Support Systems* (15), pp. 251-266.

Mendling, J., and Aalst van der, W. M. P, (2007). Formalization and Verification of EPCs with OR-Joins Based on State and Context, in: J. Krogstie, A.L. Opdahl and G. Sindre (eds.), *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, LCNS 4495, Berlin: Springer-Verlag, pp. 439-453.

Mendling, J., Moser, M., Neumann, G., Verbeek, H.M.W., Dongen van, B.F., and Aalst van der, W.M.P. (2006). Faulty EPCs in the SAP Reference Model, in: S. Dustdar, J.L. Fiadeiro and A. Sheth (eds.), *Proceedings of the 4th International Conference Business Process Management (BPM 2006)*, LCNS 4102, Berlin: Springer-Verlag, pp. 451-457.

Mendling, J., Reijers, H.A., Cardoso, J. (2007). What Makes Process Models Understandable? In: G. Alonso, P. Dadam and M. Rosemann (eds.), *Proceedings of the 5th International Conference Business Process Management (BPM 2007)*, LNCS 4714, Berlin: Springer-Verlag, pp. 48–63.

Mendling, J., Verbeek, H.M.W., Dongen van, B.F., Aalst van der, W.M.P. , Neumann, G. (2008). Detection and Prediction of Errors in EPCs of the SAP Reference Model, *Data and Knowledge Engineering* (64), pp. 312-329.

Miller G. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review*, 63 pp. 81-97.

Newell A, Simon HA. (1972) *Human Problem Solving*. Englewood Cliffs, NJ: Prentice Hall

Nielsen J. (1994) Estimating the number of subjects needed for a thinking aloud test. *International Journal of Human-Computer Studies,* 41(3), pp. 385-397.

OMG (Ed.), (2006). *Business Process Modeling Notation (BPMN) Specification*, Final Adopted Specification, dtc/06-02-01, Object Management Group.

F. Paas, J. E. Tuovinen, H. Tabbers, P. W. M. V. Gerven. (2003) Cognitive Load Measurement as a Means to Advance Cognitive Load Theory. *Educational Psychologist*, 38(1), pp. 63-71.

F. Paas, A. Renkl, J. Sweller, (2004) Cognitive Load Theory: Instructional Implications of the Interaction between Information Structures and Cognitive Architecture, *Instructional Science* 32, **pp.** 1–8.

Parsons J. & Wand, Y. (2008). Using Cognitive Principles to Guide Classification in Information Systems Modeling, *MIS-Quarterly,* 32(4), pp. 839-868.

Petri, C.A. (1962). *Kommunikation mit Automaten. Bonn: InstitutfürInstrumentelleMathematik*, Communication with Automata, English translation in Technical Report RADC-TR-65--377, Vol.1, 1966, New York: Griffiss Air Force Base.

J. Pinggera, P. Soffer, S. Zugal, B. Weber, M. Weidlich, D. Fahland, H. A. Reijers, J. Mendling. (2012) Modeling styles in business process modeling. In Proc. BPMDS '12, pp. 151-166.

Recker, J., Rosemann, M., Green, P., Indulska, M., (2011). Do ontological deficiencies in modeling grammars matter? *MIS Quarterly* 35(1)*, 57-79.*

Reijers H. A., Limam S., van der Aalst W.M.P., (2003). Product-Based Workflow Design, *Journal of Management Information Systems*, 20(1), pp. 229-262.

Reijers H.A & Mendling J. (2011). A Study into the Factors that Influence the Understandability of Business Process Models, *IEEE Transactions On Systems, Man, And Cybernetics – Part A*, 41(3), pp. 449-462.

Rittgen, P. (1999). From Process Model to Electronic Business Process, *European Conference on Information Systems ECIS 1999,* Copenhagen Business School, Copenhagen, Denmark, pp. 616-626.

Rosemann, M., Recker, J., Indulska, M., Green, P. (2006). A study of the evolution of the representational capabilities of process modeling grammars, in: E. Dubois, K. Pohl (eds.). *Proceedings of the 18th Conf. of Advanced Information Systems Engineering - CAiSE 2006*, LNCS 4001, Berlin: Springer-Verlag, pp. 447-461.

Russell, N. C., ter Hofstede, A.H.M., Aalst van der, W.M.P. , Mulyar, N. (2006). *Workflow Control-Flow Patterns: A Revised View*, BPM Center Report BPM-06-22, BPMcenter.org.

Santos S. P. Jr., Almeida J. P. A., Guizzardi G. (2010). An Ontology-Based Semantic Foundation for ARIS EPCs, *SAC'10,* Sierre, Switzerland, pp. 124-130.

Savelsbergh, E.R., deJong, T., Ferguson-Hessler, M.G.M. (1998). Competence-related differences in problem representations. In M. van Sommeren, P. Reimann, T. deJong, & H. Boshuizen (Eds.), The role of multiple representations in learning and problem solving, pp. 262-282. Amsterdam: Elsevier.

Simon, H. A. (1981) The Sciences of the Artificial, 2nd edition. MIT Press, Cambridge, MA.

Soffer, P., Kaner, M., Wand, Y. (2010). Assigning Ontology-Based Semantics to Workflow nets, *Journal of Database Management*, 21(3), pp. 1-35.

Soffer, P., and Wand, Y. (2004). Goal-driven Analysis of Process Model Validity, in: A. Persson, J. Stirna (eds.), *Advanced Information Systems Engineering (CAiSE'04)*, LNCS 3084, Berlin: Springer-Verlag, 521-535.

Soffer, P., & Wand, Y. (2005). On the Notion of Soft Goals in Business Process Modeling, *Business Process Management Journal* 11(6), pp. 663-679.

Soffer, P., and Wand, Y. (2007). Goal-driven multi-process analysis, *Journal of the Association of Information Systems* 8(3), 175-203.

Soffer, P., Wand, Y., Kaner, M. (2007). Semantic analysis of flow patterns in business process modeling, in: G. Alonso, P. Dadam and M. Rosemann (eds.), *Proceedings of the 5th International Conference Business Process Management (BPM 2007)*, LNCS 4714, Berlin: Springer-Verlag, pp. 400-407.

Sonnenberg, C., & vom Brocke, J. (2012). Evaluations in the Science of the Artificial - Reconsidering the Build-Evaluate Pattern in Design Science Research. *DESRIST 2012, pp.* 381-397.

Strauss A., &Corbin J. (1998) Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, Sage Publications

Vanderfeesten I., Reijers H. A., Mendling J., van der Aalst W.M.P. and Cardoso J. (2008). On a Quest for Good Process Models: The Cross-Connectivity Metric, in Bellahsène Z. and Léonard M. (eds.) *Advanced Information Systems Engineering (CAiSE'08)*, LNCS 5074, Berlin: Springer-Verlag, pp. 480-494.

Venable, J. R., Pries-Heje, J., Baskerville, R. (2012). A Comprehensive Framework for Evaluation in Design Science Research. *DESRIST 2012, pp.* 423-438.

Wand Y., & Weber R.,(1990) An Ontological Model of an Information System, *IEEE Transactions on Software Engineering*, 16(11), 1282-1292.

Wand, Y., & Weber, R. (1993). On the Ontological Expressiveness of Information Systems Analysis and Design Grammars, *J. of Information Systems* (3), 217-237.

Wand, Y., & Weber, R. (1995). Towards a theory of deep structure of information systems, *Journal of Information Systems* (5:3), pp*.* 203-223.

Wolf, I., & Soffer, P. (2014, January). Supporting BPMN Model Creation with Routing Patterns. In *Advanced Information Systems Engineering Workshops*(pp. 171-181). Springer International Publishing.

# Appendix 1: Merge behaviors and their completeness

| Table A1: All merge combinations | | | | | | | |
|---|---|---|---|---|---|---|---|
| | First event: domain arrives at merge | | | | | Second event: arrival of | |
| | A | | B | | Both together | A | B |
| State of domain<br>Case Number | B | C | A | C | C | C | C |
| 1 | P | U | P | U | U | | |
| 2 | P | U | P | U | S | | |
| 3 | P | U | P | S | U | U | |
| 4 | P | U | P | S | U | S | |
| 5 | P | U | P | S | S | U | |
| 6 | P | U | P | S | S | S | |
| 7 | P | U | S | S | U | | |
| 8 | P | U | S | S | S | | |
| 9 | P | U | S | U | U | | |
| 10 | P | U | S | U | S | | |
| 11 | P | S | P | U | U | | U |
| 12 | P | S | P | U | U | | S |
| 13 | P | S | P | U | S | | U |
| 14 | P | S | P | U | S | | S |
| 15 | P | S | P | S | U | U | U |
| 16 | P | S | P | S | U | S | U |
| 17 | P | S | P | S | U | U | S |
| 18 | P | S | P | S | U | S | S |
| 19 | P | S | P | S | S | U | U |
| 20 | P | S | P | S | S | S | U |
| 21 | P | S | P | S | S | U | S |
| 22 | P | S | P | S | S | S | S |
| 23 | P | S | S | S | U | | U |
| 24 | P | S | S | S | U | | S |
| 25 | P | S | S | S | S | | U |
| 26 | P | S | S | S | S | | S |
| 27 | P | S | S | U | U | | U |
| 28 | P | S | S | U | U | | S |
| 29 | P | S | S | U | S | | U |
| 30 | P | S | S | U | S | | S |
| 31 | S | S | P | U | U | | |
| 32 | S | S | P | U | S | | |
| 33 | S | S | P | S | U | U | |
| 34 | S | S | P | S | U | S | |
| 35 | S | S | P | S | S | U | |
| 36 | S | S | P | S | S | S | |
| 37 | S | S | S | S | U | | |
| 38 | S | S | S | S | S | | |
| 39 | S | S | S | U | U | | |
| 40 | S | S | S | U | S | | |
| 41 | S | U | P | U | U | | |
| 42 | S | U | P | U | S | | |
| 43 | S | U | P | S | U | U | |
| 44 | S | U | P | S | U | S | |
| 45 | S | U | P | S | S | U | |
| 46 | S | U | P | S | S | S | |
| 47 | S | U | S | S | U | | |
| 48 | S | U | S | S | S | | |
| 49 | S | U | S | U | U | | |
| 50 | S | U | S | U | S | | |
| U: unstable; S: stable; P: proceed | | | | | | | |

**Lemma:** Table A1 enumerates all possible merge behaviors in a binary-decomposable domain.

**Proof**: We show by combinatorial considerations that all possibilities were listed.

1. If both sub-domains are active, three possibilities exist for a first merge event:
a. A arrives first. b. B arrives first. c. Both arrive at the same time.
Since the sub-domains behave independently prior to the first event, these three possibilities are independent. Hence, we can calculate the total number of possible decisions by multiplying the number available for each type of first event.
2. Consider one sub-main arriving first. There are four possible decisions:
   a. C: not activated, the other sub-domain continues independently.
   b. C: not activated, the other sub-domain stops being independent.
   c. C: activated, the other sub-domain continues.
   d. C: activated, the other sub-domain stops being independent.
3. Only in case 2a, a second relevant event will occur, with two possible decisions:
   a. C: activated. b. C: not activated.

Hence, for each event where sub-domain arrives first, there are five cases.

4. Since we allow each sub-domain to arrive first, there are 25 combinations (5 when A is first x 5 when B is first).
For each combination, there exist two possibilities: 'continue on both arriving' or 'stop on both arriving'. Hence, we obtain 50 combinations.

We now reduce the list of merge behaviors (Table A1) applying two considerations.

First, symmetric cases with respect to sub-domains A and B are combined. The sub-domains are named $X_1$ and $X_2$, where $X_1,X_2 \in \{A,B\}$, $X_1 \neq X_2$. The newly organized cases are shown in Table A2, still linked to their numbers of Table A1.

Next, we add the following requirement; the catalog will include only merge behaviors where *process continuation is assured*. For a given merge behavior, whether a process can continue or not might depend on which sub-domains are active prior to the merge. For example, assume the chosen merge behavior is to stop when both sub-domains reach the merge together. In such cases, process continuation cannot be assured. This possibility can only materialize when both domains are active. Hence, it is still possible to choose it when only one sub-domain is active. However, for some behaviors the process cannot be guaranteed to continue independently of which sub-domains are active prior to the merge. For example, the process will stop on either A or B arriving first, and will continue if both arrive together (case 18 in Table A1). Since co-arrival of both sub-domains cannot be guaranteed, process continuation cannot be assured. We eliminate these cases.

The last three columns in Table A2 present an analysis of continuation certainty for the merge cases listed in the table. Each case is related to three possibilities: whether both sub-domains are active, whether only one is active, but not a specific one, or whether only a specific one is active. Each case is marked as "+" for certain continuation or "–" for continuation that cannot be assured. When continuation cannot be assured, there is no process enactment prior to the merge that assures continuation. The case is colored grey, and will be eliminated in the following step of the analysis.

| Table A2: Merge cases – continuation analysis | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **First event: domain arrives at merge** | | | | | **Second event** | **Certain continuation** | | |
| | **$X_1$** | | **$X_2$** | | **Both together** | | Both domains are active | Only one is active | Only a specific one is active |
| **State of domain**<br>**Case number** | **$X_2$** | **C** | **$X_1$** | **C** | **C** | **C** | | | |
| 1 | P | U | P | U | U | | + | + | + |
| 2 | P | U | P | U | S | | - | + | + |
| 3, 11 | P | U | P | S | U | U | + | - | + |
| 4, 12 | P | U | P | S | U | S | - | - | + |
| 5, 13 | P | U | P | S | S | U | - | - | + |
| 6, 14 | P | U | P | S | S | S | - | - | + |
| 7, 31 | P | U | S | S | U | | - | - | + |
| 8, 32 | P | U | S | S | S | | - | - | + |
| 9, 41 | P | U | S | U | U | | + | + | + |
| 10, 42 | P | U | S | U | S | | - | + | + |
| 15 | P | S | P | S | U | U | + | - | - |
| 16,17 | P | S | P | S | U | S / U* | - | - | - |
| 18 | P | S | P | S | U | S | - | - | - |
| 19 | P | S | P | S | S | U | - | - | - |
| 20, 21 | P | S | P | S | S | S / U | - | - | - |
| 22 | P | S | P | S | S | S | - | - | - |
| 23, 33 | P | S | S | S | U | U | - | - | - |
| 24, 34 | P | S | S | S | U | S | - | - | - |
| 25, 35 | P | S | S | S | S | U | - | - | - |
| 26, 36 | P | S | S | S | S | S | - | - | - |
| 27, 43 | P | S | S | U | U | U | + | - | + |
| 28, 44 | P | S | S | U | U | S | - | - | + |
| 29, 45 | P | S | S | U | S | U | - | - | + |
| 30, 46 | P | S | S | U | S | S | - | - | + |
| 37 | S | S | S | S | U | | - | - | - |
| 38 | S | S | S | S | S | | - | - | - |
| 39, 47 | S | S | S | U | U | | - | - | + |
| 40, 48 | S | S | S | U | S | | - | - | + |
| 49 | S | U | S | U | U | | + | + | + |
| 50 | S | U | S | U | S | | - | + | + |

* The behavior in the second event depends on which sub-domain is involved. For one the continuing domain will not be activated, and for the other it will be activated. See Table 3.

Finally, we identify the set of eight generic possible merge behaviors by the following considerations. All cases where continuation can be assured (Table A2) and the same decisions are made when either sub-domain arrives at the merge and are combined into one generic merge behavior. For each possible behavior we consider the process enactments (namely, what *actually* occurs prior to the merge) for which it can guarantee process continuation. If, for example, process continuation can only be assured when only one of the sub-domains is active, then what might happen to the other sub-domain is irrelevant. The cases that differ only on the decision regarding the irrelevant domain are combined. Examples are cases where the other sub-domain is allowed to proceed, or is stopped under “immediate continuation with mutual blocking,” since this behavior is guaranteed to continue only when a single sub-domain is active. The eight generic cases obtained in this way are listed in Table A3. Their detailed descriptions with examples are provided in Table 4.

| Table A3: Combining Merge Cases into Generic Types | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1st event: domain arrives at merge** | | | | | **2nd event** | **Type Name** | **Cases** |
| $X_1$ | | $X_2$ | | both | | | |
| $X_2$ | C | $X_1$ | C | C | C | | |
| P | U | P | U | U | | Immediate continuation | 1 |
| S | U | S | U | U | | Immediate continuation with cancellation | 49 |
| P | U | S | U | U | | Immediate continuation with asymmetric cancellation | 9,41 |
| P | S | P | S | U | U | Synchronization | 15 |
| P | U | P | S | U | U | Asymmetric synchronization | 3,11 |
| S | U | P | S | U | U | Asymmetric synchronization with cancellation | 27,43 |
| * | U | * | U | S | | Immediate continuation with mutual blocking | 2,10,42,50 |
| * | U | * | S | * | | Single-sided continuation | 4, 12, 5, 13, 6, 14, 7, 31, 8, 32, 28, 44, 29, 45, 30, 46, 39, 47, 40, 48 |
| U: unstable; S: stable; P: proceed; *: does not matter | | | | | | | Case numbers refer to Table A1 |

# Appendix 2: List of Workflow Patterns used in Study 1

We used Workflow Patterns included in the following groups (http://www.workflowpatterns.com/patterns/control/index.php):

1) Basic control flow patterns – simple merge and split structures; the sequence pattern was excluded (not a split/merge structure).
2) Advanced branching and synchronization patterns – additional advanced split and merge structures. We excluded patterns identified as combinations of several basic control flow patterns, non-binary merges, and execution-related patterns.
3) Trigger patterns – Transient and Persistent. Both these patterns depend on a signal from the external environment that we interpret as a sub-domain operating independently (and hence consider these as merge patterns). However, based on this interpretation a persistent trigger is simply a recurring case of synchronization. Hence, we included the transient trigger pattern that is an asymmetric merge depending on the trigger activation.

Other groups of patterns were excluded for one of the following reasons:

1) They address multiple instances or multiple cases;
2) They exhibit execution-related behaviors;
3) They include combinations of several splits and merges that are redundant according to our interpretation.

The list of patterns used for our evaluation includes:

### *Splits*

*Parallel Split*: both branches are active (AND-split).
*Multi-Choice*: each branch, and both combined, can be active (OR-split).
*Exclusive Choice:* only one branch can be active (XOR-split)

***Merges***

*Synchronization.* Both branches are active, and the first to arrive waits for the second to continue.

*Simple Merge.* Only one branch is active and the process continues when it arrives.

*Structured Synchronizing Merge.* When both branches are active, the first to arrive waits for the second to continue. When only one branch is active, the process continues when it arrives.

*Multi-Merge.* When both branches are active, each of them activates the merge.

*Structured Discriminator.* When both branches are active, the process continues when the first one arrives. The other branch continues to completion.

*Cancelling Discriminator.* When both branches are active, the process continues when the first one arrives. The other branch stops.

*Transient trigger.* Branch A (task instance) waits for branch B (trigger) to continue. If a trigger is given (branch B arrives) before branch A has arrived, the merge is not activated.

# Appendix 3: Situations in the experimental materials

Situation Descriptions:

- Situation 1: A process of handling machine failure. When failure in a machine part is identified, in-house maintenance tries to fix it. If it is very urgent to have the machine operational, a replacement part may be ordered from the supplier. If the part is fixed before the ordered one arrives, the order is cancelled. If the ordered part arrives before the part is fixed, it is installed, but fixing will be continued and the fixed part may be saved for future needs.
- Situation 2: In a purchasing department. Buyers always seek quotations from a preferred supplier, but they also seek quotations from alternative supplier/s for possibly better quotes. If the preferred supplier's quotation arrives first, the buyer immediately prepares an order. The quotations from the alternative suppliers are saved for the future suppliers' ranking. If the first quotation to arrive is from the alternative supplier, the buyer waits for the preferred supplier's quotation before deciding from whom to order.
- Situation 3: Product development problems. Two engineering teams deal with resolving various product development problems. Sometimes a problem is handled by one of two teams, and sometimes both teams work on the problem. Fixing the problem should be based on integration of all solutions proposed by the teams.
- Situation 4: Customer claim to an insurance company. To reduce waiting time, a customer claim to an insurance company is handled by two different claim representatives. They both use a standard procedure for documenting and handling the claim. After one of them completes the procedure, the claim is closed and the second representative stops. The customer is informed.
- Situation 5: A transportation company fulfills shipment orders. Some of the shipments include products that require refrigeration. A refrigerating truck can handle all kinds of shipment, but a regular truck cannot be used for the refrigerated products. There are many regular trucks and only one refrigerating truck. Shipment orders cannot be split to several trucks. If it is known in advance that a shipment order for refrigerated products will arrive, then the refrigerating truck should be reserved for it. If a refrigerating truck is not available when an order for refrigerated products is due, the shipment cannot take place. (Note: the

students were asked to refer to both possibilities; whether the refrigerating truck was reserved or not).

- Situation 6 (used in Study 2 only): Report of sales figures. A company that sells both pharmaceutical and cosmetic products reviews its sales every quarter. In this review each product manager prepares a report of sales figures. For products that have been marketed for less than two years, a consumers' evaluation report (for cosmetics) or physicians' evaluation report (for pharmaceuticals) is prepared in parallel to the sales report. If the sales report is ready before the (relevant) evaluation report, a presentation is prepared based only on this report, and the preparation of the evaluation report is abandoned. Otherwise, if the (relevant) evaluation report is ready before the sales report, the presentation is prepared when the sales report is ready too, and includes figures from both.

| Table A4: Understanding task questions and answers (cases 2-6) | | |
|---|---|---|
| **Situation** | **Statement (true or false)** | **Expected answer** |
| 2 | The preferred supplier's quotation arrived yesterday but still the decision (supplier selection) has not been made. | False. If the quotation arrives from the preferred supplier, the buyer proceeds to prepare an order. |
| | The preferred supplier's quotation arrived, and the additional inquiry (alternative suppliers) is not needed so it is not made. | False. Quotations are requested from the preferred and additional suppliers. |
| | If there are two quotations, two orders are made. | False. One supplier is selected. |
| | The order was placed although one of the quotations has not arrived yet. | True, if the quotation from the preferred supplier arrives first. |
| | There is a possibility that the worse quotation (higher price) will be accepted. | True, if the preferred supplier's quotation arrives first. |
| 3 | The problem was resolved through the solution proposed by one of the teams. | True, if only one team worked on the problem. |
| | The problem was resolved through the solutions proposed by both teams. | True, if two teams worked on the problem. |
| | The problem was resolved through the solution proposed by one of the teams, and then solution of the second team was proposed. | False. Solutions are integrated if two teams are involved. |
| | The solution proposed by one of the teams was rejected. | False. The overall solution is based on the integration of all the solutions proposed. |
| | The solution was proposed by the team that finished the work. | True, if one team worked on the problem. |
| 4 | The claim was closed because one of the representatives finished his work. | True. When the first one finishes the procedure, the claim is closed. |
| | The claim is open as only one of the representatives finished his work. | False. When the first one finishes the procedure, the claim is closed. |
| | The customer is informed and one of the representatives continues his work. | False. He stops. |
| | Both representatives close the claim. | True, if they finish together[18]. |
| | The representative to be responsible for the claim handling is selected from two possible representatives. | False. Both representatives deal with each claim |
| 5 | The products are ready for shipment for several days but the shipment cannot be performed as there is no regular truck. | False. There are many regular trucks |
| | The shipment cannot take place as refrigerating truck was not reserved. | True. If the truck was not reserved, it may not be available. |
| | The shipment cannot take place as the refrigerating truck is occupied by another shipment. | True. If the truck was not reserved, another delivery can be transported by it. |
| | The shipment is transported by a regular truck. | True. If it is a regular shipment. |
| | Some of the products that require refrigerating are shipped by the refrigerating truck, others by a regular one. | False. A shipment cannot be split. |
| 6 | The presentation is prepared based on the sales report and the physicians' evaluation. | True. If physicians' evaluation was completed first (for pharmaceuticals). |
| | The presentation is prepared based on the consumers' and the physicians' evaluation. | False. It can be either but not both. |
| | The product is marketed over two years. The consumers' evaluation is ready, but the sales report is still awaited. | False. If the product is marketed over two years, consumers' evaluation is not needed. |
| | The product is marketed for less than two years. The presentation is prepared based on the sales report only. | True, if the sales report was ready first. |

[18] If the answer given was "False," explaining the probability for finishing at the same moment is zero, the answer was also considered correct.