WORKFLOW SEARCH SPACE REDUCTION — A MODEL DRIVEN APPROACH

Mati Golani
Ort Braude College – Israel
P.O. Box 78, Snunit 51, Karmiel, 21982, ISRAEL
e-mail: matig@braude.ac.il

Abstract: To date, the ability of a business process designer to produce a solid, well-validated workflow models is limited, especially since all necessary scenarios that need to be covered by the workflow are hard to predict. Workflow management systems (WfMSs), serving as the main vehicle of business process execution, should recognize those limits, and increase its support to designers in this task. One aspect of such assistance is in exception handlers generation. In this paper we propose a model language enrichment for expressing workflow semantics, in the context of alternative solutions, within the process model. Thus, enabling the designer to state which possible alternatives and their applicability to changing execution paths states. Using this enrichment, an inference algorithm can efficiently find an adequate alternative. The model language is used as a basis for a design tool and an execution environment, which semi-automatically generates exception handlers, resulting, due to a reduced search space, in a smaller set of exceptions for the designer/user to choose from.

AMS Subject Classification: 03H10, 90A16
Key Words: workflow model, workflow management systems, business process

1. Introduction

The ability of a business process designer to produce solid, well-validated workflow models is limited as it is difficult to predict all various scenarios, needed for the process model. Workflow management systems (WfMSs), serving as the main vehicle of business process execution, should recognize these limits and
become more case oriented, assisting the designers in this task. One aspect of such assistance is in exception handlers generation. To illustrate this point, we next present an example, involving Web services. First, we observe that there is no existing mechanism that can solve the designer’s effort in modeling alternative paths. Web services, for example, merely provide syntactic information regarding their input, output and processing logic, through standards such as WSDL [19]. Usually, such descriptions fail to convey constraints and restrictions. Web services choreography does not refer to alternative paths affinity. This means that we cannot tell whether one path can be executed as a result of exception, while some Web service was invoked on an alternative one. Modeling using Web services, therefore, is likely to make the validation of workflow models more difficult [9], and more exceptions at run-time are to be expected. Efficient exception handling is a fundamental component of WfMSs and is critical to their successful implementation in real-world scenarios [3, 11] and has great impact on system performance.

Designing efficient exception handlers – specifications of exception handling processes – is not a simple task. By their very nature, exceptions are rare events that do not enjoy the advantages of common processes, which are easily programmed with much expert information injected into them. Thus, exception handlers may well be ill-designed, affecting both the correctness and the efficiency of the process. Avoiding their design altogether is also not a valid option. During runtime, process operators observe a narrow perspective of the process, and given an exception, will not have sufficient information to effectively manage it. In those cases in which the operator mitigates the exception, the solution may be neither optimal nor effective.

This knowledge is, needless to say, not intuitive, but rather quite complicated to model using the conservative constructs. However it looks like a classical case for alternative path exception handler (i.e. rollback and execute another direction from a visited Xor split activity). Therefore, there is a demand for an extended model that considers such alternatives, of which some are state based. Such a model can assist the designer in generating more accurate exception handlers, and a whole more accurate process model.

In this paper we propose a model enrichment for expressing alternative execution paths, using real world semantics. By augmenting the process model with additional meta-data, the designer is capable of stating applicable alternative paths, with respect to changing world states and conditions. Once the process model is defined, a tool may use the additional information in order to assist both the designer during design-time, or the user during runtime, with
relevant exception handlers. This extension provides a more complete reflection of reality. When referring to auto-generated exception handlers, it reduces the number of illogical solutions, and generates a smaller set of exceptions for the designer/user to choose from. Our objective is to minimize the necessary user intervention when handling an exception in the business process execution. Therefore, we suggest methods to use the world knowledge and process model to reduce the decision space of the user.

The rest of the paper is organized as follows. In Section 2, we provide the preliminary constructs on which we base our model regarding process modeling and exception definition. Then, Section 3 presents the extension to the model, to address the relations and interactions between alternatives. Section 4 introduces the various semantic options between alternatives in a Xor split activity, such as the conservative mutual Xor, asymmetric alternative path execution, and conditional alternative path execution. We also discuss the quality metrics in terms of a minimal change. In Section 5 we give an illustrated example. Section 6 reviews related work on this topic, and finally we conclude in Section 7.

2. Workflow Model

In this section we define basic constructs in workflow graphs, to be used later in the paper. The classification of workflow constructs is not new and has been discussed in various works (e.g., [2]).

There are various ways to describe a workflow process model. Workflows define a business process in terms of activities (also called actions or tasks). Activities, together with temporal constraints on execution ordering define a business process [18]. A workflow model can be graphically described as a workflow net [1] for example, or ADEPT WSM net [17]. In what follows, and following the WFMC definition (in interface 1) of full-blocked workflows, we assume the use of well structured processes.

A workflow model can be described as a graph (ADEPT WSM net) $G(V, E)$ $(V = (V_a \cup V_d); E = (E_c \cup E_d))$, where $V_a$ is a set of activities, $V_d$ is a set of data parameters, $E_c$ is a set of control edges, and $E_d$ is a set of data edges. For simplicity, whenever possible, we will refer to the reduced graph $G' = G(V_a, E_c)$. excluding data flows

An activity $a$ in $V_a$ has in-degree($G', a$) incoming edges and out-degree($G', a$) outgoing edges. Whenever it becomes clear from the context, we eliminate
the graph reference and refer to $in-degree(a)$ and $out-degree(a)$. A path in $G$ is a set of activities such that any two consecutive activities on the path are connected by an edge in $E_c$. We denote a path from $a_i$ to $a_j$ by $(a_i, \ldots, a_j)$. The length of a path $(a_i, \ldots, a_j)$ (denoted $length(a_i, \ldots, a_j)$) is the number of edges in $(a_i, \ldots, a_j)$. Finally, $Minlength(a_i, a_j)$ is the length of the shortest path in $G$ that starts at $a_i$ and ends at $a_j$.

We next define two graph constructs, namely splits and joins, based on the Workflow Management Coalition standard [18]. A Xor split is a node (activity) with multiple outgoing edges ($out-degree(a)>1$), only one of which can be followed in the execution flow. The decision as to which edge to follow is based on the satisfaction of mutually exclusive conditions that are typically associated with the outgoing edges. Let $c_{a,a'}$ be a DNF (Disjunctive Normal Form) Boolean statement with a set of variables $Var(c_{a,a'})$ that must be satisfied in order to pass from activity $a$ to activity $a'$. Activity 3 in Figure 1 is an example of a Xor split. Each Xor split $a$ is associated with a Xor join (e.g., activity 11 in Figure 1), an activity common to all paths that start from $a$. During runtime, when reaching $a$, the workflow engine evaluates the conditions on each of $a$’s outgoing edges, and continues the execution along the edge whose associated condition is satisfied. The Xor join activity acts as a synchronization point in the execution.

An And split is a node with multiple outgoing edges whose execution flow follows all outgoing edges by parallel threading. Threads of an And split $a$ need to be synchronized at an And join, which is also a node in the graph that is common to all paths that start from $a$.

**Definition 1.** (Xor Split Point) Let $G^e = (V, E)$ be a workflow graph, and
a be an activity in V. A Xor split point of a is a Xor split aᵢ with a Xor join aⱼ such that aᵢ is a predecessor of a and aⱼ is a successor of a.

**Definition 2.** (NXSP) Nearest Xor split point of a \((NXSP(a))\), is a Xor split point of a, aᵢ, which satisfies that any other Xor split point of a \((aⱼ)\) is also a Xor split point of aᵢ.

And split point and NASP are similarly defined. Using the basic definitions given above, we now define blocks in a graph. Let \(G' = (V, E)\) be a workflow graph and let aᵢ be a Xor split and aⱼ be the Xor join associated with aᵢ. A Xor block of aᵢ is a subgraph of \(G'\) induced by the nodes of all paths \((aᵢ, ..., aⱼ)\) in \(G'\). Similarly, given an And split aᵢ and the associated And join of aᵢ, aⱼ, an And block of aᵢ is a subgraph of \(G'\) induced by the nodes of all paths \((aᵢ, ..., aⱼ)\) in \(G'\).

Clearly, any activity a is within a Xor block defined by its Xor split point (can be null) and its associated Xor join. In particular, a is within a Xor block defined by \(NXSP(a)\) and its associated Xor join.

**Definition 3.** (Alternative Paths) Let \(G' = (V, E)\) be a workflow graph with a sink f, and let \(P₁ = (aᵢ, ..., aⱼ)\) and \(P₂ = (aᵢ, ..., aₖ)\) be paths in \(G'\). \(P₁\) is an alternative to \(P₂\) (and vice versa) if the following four conditions hold:

1. \(aᵢ\) is of type Xor split.
2. There is no activity a in \(V\{aᵢ\}\) such that a is in \(P₁\) and a is also in \(P₂\).
3. Any path \((aⱼ, ..., f)\) in \(G'\) does not include an activity in \(P₂\).
4. Any path \((aₖ, ..., f)\) in \(G'\) does not include an activity in \(P₁\).

It is worth noting that \(P₁\) and \(P₂\) share a common initial activity aᵢ. As an example, consider the alternative paths \((3,4,13,5,6,10,14)\) and \((3,12,8,9)\) in Figure 1. Note that the paths \((5, 6)\) and \((5, 7)\) are not alternative paths, since activity 5 is not of type Xor Split (both activity 6 and activity 7 are part of the same And block).

The importance of Xor blocks in our analysis is related to the ability to provide an alternative paths analysis. In Figure 1, the Xor Block includes the entire graph save activitis \(\{0,1,2\}\), and thus an alternative path for any activity (excluding activity 11) will start from activity 3. Therefore, once activity 6 fails, the Xor Block to which activity 6 belongs allows an alternative execution, using the paths that contains activities \(\{3,12,8,9\}\).

We next discuss the normalization of Xor and And blocks. A normalized (Xor or And) block is a block in which neither the outgoing edges of the split
activity, nor the incoming edges of the join activity, are connected to any activities outside the block. This property matches the WFMC definition (in interface 1) of full-blocked workflows. Formally,

**Definition 4.** (Normalized Block) Let $G'(V, E)$ be a workflow graph with a source $s$ and a sink $f$, and $B$ a Block (either Xor or And) with split activity $a_i$ and join activity $a_j$. $B$ is normalized if $a_j$ is on all paths $(a_i, \ldots, f)$ in $G$ and $a_i$ is on all paths $(s, \ldots, a_j)$ in $G'$.

It is easy to show that if $B$ is normalized, then $\text{out-degree}(G', a_i) = \text{out-degree}(B, a_i) = \text{in-degree}(G', a_j) = \text{in-degree}(B, a_j)$. For brevity, we refrain from presenting the algorithm for block normalization in this paper.

Given a workflow graph $G'(V, E)$, $\text{Inst}(G')$ represents an instance of $G'$. $\text{Inst}(G')$ encapsulates instance-related data, such as activity state and input/output parameter values. $\text{Inst}(G')$ is a DAG and loop constructs in $G'(V, E)$ are removed by duplicating loop blocks and re-labeling of activities.

An activity in $\text{Inst}(G')$ can be classified into one of the following states: uninitiated (yet, but on an execution path), void (on path that was not invoked), completed (finished on current path), compensated, or failed.

### 2.1. Exception Handling

An exception handler is a workflow $X(V_X, E_X)$, executed in response to an occurrence of an exception for which it was defined. Given a workflow $G(V, E)$ and an exception handler $X(V_X, E_X)$, we define an operator $\text{Apply}$ such that by applying $X(V_X, E_X)$ to $G(V, E)$ one receives a revised workflow model $G'(V', E') = \text{Apply}(G(V, E), X(V_X, E_X), v_s, V_e)$, where $\{v_s\} \cup V_e \subseteq V$. $v_s$ specifies the failing node in $G$ and $V_e$ is a set of nodes in $G$ from which the normal operation of $G$ will resume [12]. An exception handler can be schematically partitioned into two sections, namely rollback and forward stepping. A rollback section executes compensating activities and a forward stepping section activates and reactivates activities. For each exception handler we can define three reference activities. We denote by $a_{sr}$ a start activity, the failing activity. $a_{sp}$ is a stop activity, the activity where the control is returned to the original process. Finally, a target activity ($a_{tr}$) is the activity where the rollback section ends.

There are three types of activities an exception handler can use. The first type can activate activities in the workflow (for the first time), or reactivate
them. The second type invokes compensation activities, also known as undo activities and semantic rollback activities [13, 10]. A compensating activity needs to be pre-defined, is associated with a single or combined set of workflow activities, and is typically used for reversing the impact of activities that were already performed for a given instance. Lastly, an exception handler can use activities that are not defined in the workflow altogether.

In [12] two types of exception handlers were presented. A repeat activation exception handler. Such an exception handler attempts to repeat the activation of a subgraph of a workflow model by first applying compensating activities to the part that was already activated, followed by reactivation of activities. The second type is denoted an alternative path exception handler which was first introduced in [11]. Alternative path exception handlers combine the use of compensating activities, reactivation of activities and first-time activity activation. For an alternative path exception handler a_sp is always a Xor split point which is a direct or predecessor NXSP for the referred activity [11].

Another distinction is between actual and logical execution in an exception handler. An actual activation of an activity a involves the invocation of a routine associated with a or performing a new work item in an item list of some role in the organization. A logical activation of a requires only recording its activation in the WfMS without actually activating it. C (a) is set to 0 whenever a requires only a logical activation. If an actual activation is involved, then C(a) is assigned with its full cost. The cost of exception handler X is the sum of costs of all activities in X.

3. Modeling Alternative Flows

In [11] it was stated that the user may be invoked with too many and infeasible alternatives, while a relaxation can somehow address this issue. A process model reflects real world context. Exception, by its nature, is context related. In the following section, we extend and study further the model semantic context in order to minimize the user invocation, and infer better and faster exception handling solutions. This is done by a classification of the process model context. Each Xor split activity is augmented with additional meta-data, enabling the modeling of the following classification.

— Global class (Stateless) – can be evaluated by inspecting the model regardless a specific instance data.
— Instance based class – requires instance data for exception feasibility
evaluation

— Direction related — dependent on the actual path taken from the Xor split activity

— State related — dependent on the non/activation of a specific activity in the path taken from the Xor split activity.

We continue by using a predicate model as a formal extension meta-data to the process model in order to enrich the process language. Each decision edge \( e_{ij} \) has the following attached structure: \( Dec = (\Phi_{ij}, I_{ij}, c_{ij}) \).

— \( I = \{\tau_1, \tau_2, \ldots\} \) — a set of invariants, one for each \( e_{ik} : k \neq j \) in a CNF structure for evaluating \( \tau_{ik} \) as a condition for executing \( e_{ij} \) as an alternative path.

— \( \Phi_{ij} = p_1 \land p_2 \land \ldots \) — a logical formula in CNF structure representing the routing criteria decision.

— \( c : I \rightarrow price \) — a pricing function.

\( W \) — a CNF structured formula \( p_{w1} \land p_{w2} \land \ldots \) that represents the world’s state. \( \forall \{\text{parameter, operator, value}\} \in p_{wi} : \text{parameter} \in V_d, \) each valid state \( W \) is evaluated to true. \( W \) used to determine the solution space for which a formula composed from \( \Phi \) and \( I \), is evaluated for available solutions, to be discussed in Section 4.

4. Alternative Related Knowledge Modeling

Given a workflow process \( G \) that includes some Or split activity \( a_x \), the semantic reasoning of the process domain may provide us with different meaning of relating to this activity \( a_x \). The conservative approach claims that the two Xor activity branches are mutually exclusive. That means that if the process instance took one branch, it cannot take the other. Consider the example in Figure 1, and the Xor split activity 3. The actual branching direction is taken according to the value of \( c \). We also assume that this value is not system dependent but rather a user input (e.g. a controllable choice [5]). An example for such a case may be a medical patient registration that includes different pregnancy tests, while activity 3 checks the pregnancy status of the patient. In this case there is no real alternative meaning for the two directions. Either the patient is pregnant or not.

In that case, \( \exists i \in I_{3-4} : i = \Phi_{3-12} = e2 \). The compound formula is \( c1 \land c2 = (c = 1) \land (c = 2) \) which is evaluated to false. Thus, for a given a world state \( W \),
and an execution that was carried out using $e_{i-j}$, there is no solution $W^{desired}$ that satisfies the execution via $e_{i-k}$. Formally put:

\[ W \Rightarrow \not\exists W^{desired} \models (t_{i-j} \land \Phi_{i-j}) \]

\[ W \Rightarrow \not\exists W^{desired} \models (t_{i-k} \land \Phi_{i-j}) \]  \hspace{1cm} (1)

The $\land$ operator is used in order to produce a formula that takes the values in the original path (left hand side proposition i.e. $t_{i-j}$) but ensures its validity on the alternative one (right hand side proposition).

A more forgiving option is providing an execution over the alternative path [11]. Here, using the alternative path exception handler, though the process has been executed on one path in can be compensated and continue over another path. In this case the $e$ referring to the other path is an empty set. Referring again to our example, that means, that while executing activity 12 the process may in some cases rollback to activity 3 and continue its execution in the path going through activity 4. An example for such a case is an ordering process, were the Xor split differ between gold customer and regular customer. In case of some malfunction in the gold customer path, the order can be always executed as a regular customer (given the inheritance nature with some degree of compensation).

Formally put

\[ W \Rightarrow \exists W^{desired} \models (t_{i-k} \land \Phi_{i-j}) = F. \]

And in our example there is a solution $W^{desired}$ which satisfies $F$.

\[ [t_{3-12} : (t_{3-12} \in I_{3-4})] = \emptyset \Rightarrow t_{3-12} \land \Phi_{3-4} = \Phi_{3-4} = \{ (c = 2) \} = F, \]

\[ \{ W^{desired} = (c = 2) \} \models F. \]  \hspace{1cm} (2)

4.1. Asymmetric Alternative Path Execution

We would like to draw the readers’ attention that the above formula does not state just the routing conditions (as appear in Figure 1), but also the set of propositions that can hold along the left side propositions set. Practically this means, that a process with original value of $c = 2$ may be executed on $e_{3-4}$ since modification of $c$ in this direction is allowed.

On the other hand, in our scenario, the opposite direction is not valid (i.e. a regular customer treated as a gold one). We can enforce it by

\[ (t_{3-4} \in I_{3-12}) \land (\Phi_{3-12})^{always} = false. \]
Formally put
\[
(t_i \in I_i \land \Phi_i) \land \Phi_i \leftarrow \textit{always} \quad \text{false},
\]
\[
(t_i \in I_i \land \Phi_i \leftarrow \textit{true}.
\]

The usage of equation (3) is presented in Algorithm 1.

**Algorithm 1** Semantic inference execution algorithm

1. **Input:** Graph \( G' \), \( \text{Inst}(G) \) - Instantiation, \( a_j \) - alternative path candidate activity.
2. **Output:** approval - a Boolean value map that represents the approval for each alternative.
3. **Process:**
   4. //execute over the possible alternative paths from \( a_j \)
   5. set \( e_{j,k} \) to be the activated edge on the original process
   6. for each \( e_{j,l} \) where \( e_{j,l} . \text{history}=\text{false} \) do
   7. if \( \Phi_j \) then
   8. set approval\((e_{j,l})=\text{true}
   9. else
   10. set approval\((e_{j,l})=\text{false}
   11. end if
   12. end for
13. return approval

### 4.2. Conditional-Alternative Path Execution

Consider a case were a certain path may rolled back unless some specific activity is already executed or completed. Taking for example a chemical analysis execution process, which involves providing a sample \( (a_1) \), declaring the type of analysis \( (a_2 \text{ type Xor split directing to } a_3 \text{ with condition } c \text{ or to activity } a_{10} \text{ with condition } \bar{c} \) ), preparing the analysis kit \( (a_3) \), and performing the analysis on the sample \( (a_4) \). Suppose that the given activities describes a specific test (e.g. gas chromatography) that involves some modifications on the sample in activity \( a_4 \) (for preparations to the test). In a case of a desired rollback for executing another analysis, if activity \( a_4 \) is already executed, then the sample was modified. Thus, the alternative path could not be taken.

On the contrary case, where the decision is taken while activity \( a_3 \) is active, the rollback is valid and may be taken. This can be modeled as a state predicate within the propositions. This predicate is evaluated during runtime. In order
to satisfy this constraint the following clause in \( t_{e_2, 3} \) should hold
\[
c \implies (a4.activated \lor a4.completed \lor a4.aborted),
\]
\[
a4.activated \implies a4.active
\]
\[
a4.completed \implies a4.active
\]
\[
a4.aborted \implies a4.active
\]

For simplicity we classify every activity that was activated during the execution as \textit{active}, which brings us to a shorter form:
\[
c \implies a4.active.
\]

Using first order logic equivalence rules
\[
c \lor \neg (a4.active).
\]

Since we apply the conjunction of \( \iota \) and \( \Phi \) we get in result
\[
\text{AlwaysFalse} \quad \bar{c} \land (c \lor \neg (a4.active)) = (\bar{c} \land c) \lor (\bar{c} \land \neg (a4.active)).
\]

Obviously, the first section is always evaluated to false. Therefore, the second section should be always evaluated to true in order to permit a solution \( W_{desired} \) which executes along the alternative path. This means that the activation of activity \( a_4 \) is not allowed

For a given set \( t_{e_{i-1}, j} \),
\[
\forall (\neg a_k.\text{active}) \in t_{e_{i-1}, j} : a_k \text{ was not activated} \implies \text{approval}(e_{ij}) = \text{true}.
\]

For simplicity each clause \( p \) in \( I \) is fragmented into two fragments a model fragment \( f_m \), and a runtime (instance) fragment \( f_i \), where \( f = f_m \land f_i \). In the above example \( f_i = (\neg a4.active) \)
\[
\exists W_{desired} \models f_m(p_x \in t_{i-1}\land \Phi_{i-k}). \quad (4)
\]
Equation 4 is a mandatory condition but not a satisfactory one for using the \( i \rightarrow j \) direction as an exception handler. The satisfactory condition contains in addition
\[
W_{desired} \models f_i(p_x \in t_{i-1}). \quad (5)
\]

4.3. Discussion

Once an alternative path has been established, the procedure proposed in [11] involves identifying the data items that need to be modified to allow the workflow instance to use the alternative path (dubbed, heretoafter, \textit{the change set}). For each such data item, we need to trace back the activity in which it was modified and request to modify the data item value to the new value. It would
be desirable to find the minimal change set for two main reasons. First, our aim is to avoid, to the extent possible, performing redundant extra work. Therefore, changing a single data item is preferred over changing two data items to turn the condition on the outgoing edge of the Xor node of the alternative path to be true. Second, if we are able to compute the minimal change set, we can use this information to optimize our efforts to recover from an exception. This way, we can compare the “cost” of various alternative paths, rank them in an increasing order of their cost, and try them one by one.

Although finding the minimal change set is desirable, it is not an easy task. In fact, finding the minimal change set is an NP-complete problem [4]. This can be shown by performing a reduction from the minimal set cover problem. Therefore, in our future work we shall attempt to identify good heuristics that take into account the semantics of workflows to improve the solution performance.

5. Illustrating Example

Consider Figure 2 that presents a scenario of a medical care within a hospital. In this process there are two points of decision making (namely “Decide treatment”, and “Decide medicine”). In first one, the doctor decide whether a surgery is required, or a medicine treatment is sufficient. The second activity refers to the type of medicine treatment (oral vs. iv). The conservative approach semantics means, that the various options are mutual exclusive. Suppose that the designer is introduced with some more knowledge from the domain experts (i.e. doctors):

— If an operation begun or completed, in case of some exception (reduction in the patient stability), the path of the medicine treatment can be taken.

— After a medicine was taken, performing a surgery is forbidden. If a decision to take a medicine was taken, there is still an option of performing an operation on emergency cases unless the medicine was actually applied.

— Until the medicine is applied, one can switch – in case of short inventory-from oral to iv and vice versa.

We next show how to model the extended requirements presented in Section 1. For simplicity we tag some of the activities with numbers: 1=decide_treatment; 2=check_ins; 3=decide_medicine; 4=IV_calcDosg; 5=Oral_calcDosg; 6=apply_med; 7=perform_op. In our example: Φ_{17} = \{t = surgical\}, Φ_{12} = \{t = medicine\}, Φ_{34} = \{m = IV\}, Φ_{35} = \{m = Oral\}. 
Now, let us refer to the given requirements.

— If an operation begun or completed, in case of some exception (reduction in the patient stability), the path of the medicine treatment can be taken.

\[ I_{12} = \emptyset \text{.} \]

— After a medicine was taken, performing a surgery is forbidden. If a decision to take a medicine was taken, there is still an option of performing an operation on emergency cases unless the medicine was actually applied.

\[ I_{17} = \{(t = \text{medicine}) \lor \neg(6.\text{active})\} \text{.} \]

— Until the medicine is applied, one can switch - in case of short inventory-from oral to iv and vice versa.

\[ I_{24} = \emptyset, I_{25} = \emptyset \text{.} \]

There is no requirement to refer to the “apply medicine” activity state, since it is the closing join activity of split activity 3. Thus, once this activity is activated, activity 3 is not an option for alternative path. Recall that we check the feasibility of \( I \land \Phi \) on candidate XSP activities, and activity 3 is not such one.

### 6. Related Work

As mentioned in the introduction, there are several approaches for dealing with exception modeling, while not addressing the semantic alternative availability of other paths. A compensation based rollback as described in [8], using design time specification [6, 15, 7]. A second approach is dynamic exception han-
dling generated on runtime [14]. Other works used a dynamic interaction with the user for exception handling inference [11]. In [16], the authors addressed a relevant field of automatic service composition for providing an alternative execution while specifying some shortcomings, and refer to the requirement of alternative control flow, and uncertainty in the initial state and service effect.

7. Summary and Outlook

In this paper we proposed a model enrichment for expressing real world semantics, in the context of alternative solutions, within the process model. By injecting extra meta-data into the process model, the designer is capable of stating those semantics. Once the process model is defined, a case tool may use this extended data in order to assist both the designer during design-time, or the user during runtime, with relevant exception handlers.

This extension provides a more complete reflection of reality. When referring to auto-generated exception handlers, it reduces the number of illogical solutions, and generates a smaller set of exceptions for the designer/user to choose from.

Currently we implement a prototype as a proof of concept, and intend to evaluate it on real world scenarios. Future work may include integration with a real workflow engine. Present plans are to focus on costs, and run solvers (such as SAT) in order to rate available alternatives.

References


