Open access • Proceedings Article • DOI:10.1109/EDOC.2009.24

Process SEER: A Tool for Semantic Effect Annotation of Business Process Models — Source link

Kerry Hinge, Aditya Ghose, George Koliadis

Institutions: University of Wollongong

Published on: 01 Sep 2009 - Enterprise Distributed Object Computing

Topics: Business Process Model and Notation, Business process modeling, Business process, Business process management and Process modeling

Process SEER: a tool for semantic effect annotation of business process models

Kerry G. Hinge
University of Wollongong, khinge@uow.edu.au

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

George Koliadis
University of Wollongong, gk56@uowmail.edu.au

Process SEER: a tool for semantic effect annotation of business process models

Abstract

A key challenge in devising solutions to a range of problems associated with business process management: process life cycle management, compliance management, enterprise process architectures etc. is the problem of identifying process semantics. The current industry standard business process modeling notation, BPMN, provides little by way of semantic description of the effects of a process (beyond what can be conveyed via the nomenclature of tasks and the decision conditions associated with gateways). In this paper, we describe the conceptual underpinnings, design, implementation and evaluation of the process SEER tool that supports several strategies for obtaining semantic effect descriptions of BPMN process models, without imposing an overly onerous burden of using formal specification on the analyst. The tool requires analysts to describe the immediate effects of each task. These are then accumulated in an automated fashion to obtain cumulative effect annotations for each task in a process. The tool leverages domain ontologies wherever they are available. The tool permits the analyst to specify immediate effect annotations in a practitioner-accessible controlled natural language, which enables formal specification using a limited repertoire of natural language sentence formats. The tool also leverages semantic Web services in a similar fashion.

Disciplines

Physical Sciences and Mathematics

Publication Details

Hinge, K., Ghose, A. K. & Koliadis, G. (2009). Process SEER: a tool for semantic effect annotation of business process models. Thirteenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2009) (pp. 54-63). Los Alamitos, USA: IEEE.

2009 IEEE International Enterprise Distributed Object Computing Conference

Process SEER: A Tool for Semantic Effect Annotation of Business Process Models

Kerry Hinge*, Aditya Ghose† and George Koliadis‡
Decision Systems Laboratory
School of Computer Science and Software Engineering
University of Wollongong, NSW 2522 Australia,
*Email: *kgh72@uow.edu.au - †aditya@uow.edu.au - ‡gk56@uow.edu.au*

Abstract

A key challenge in devising solutions to a range of problems associated with business process management: process life cycle management, compliance management, enterprise process architectures etc. is the problem of identifying process semantics. The current industry standard business process modelling notation, BPMN, provides little by way of semantic description of the effects of a process (beyond what can be conveyed via the nomenclature of tasks and the decision conditions associated with gateways). In this paper, we describe the conceptual underpinnings, design, implementation and evaluation of the ProcessSEER tool that supports several strategies for obtaining semantic effect descriptions of BPMN process models, without imposing an overly onerous burden of using formal specification on the analyst. The tool requires analysts to describe the immediate effects of each task. These are then accumulated in an automated fashion to obtain cumulative effect annotations for each task in a process. The tool leverages domain ontologies wherever they are available. The tool permits the analyst to specify immediate effect annotations in a practitioner-accessible controlled natural language, which enables formal specification using a limited repertoire of natural language sentence formats. The tool also leverages semantic web services in a similar fashion.

1. Introduction

The growing interest in business process management (BPM) methodologies and tools, as well as the high levels of adoption of such technologies in industry has led to a greater need for more sophisticated techniques for analysing and reasoning with business process models. Much of the analysis required for process compliance management [1], change management [2], enterprise process architectures [3] and the management of the business process life cycle [4] relies on being able to refer to the *semantics* of business processes. The current industry-standard process modelling notation, BPMN [5], as well as several other similar notations, provide a means for describing the *coordination semantics* of business processes but not the semantics of processes in terms of their *effects*. Thus a BPMN process model might require that task A must precede task B, but does not provide any indication of what is done by tasks A and B (beyond what might be implicit in their nomenclature), i.e. their effects. We are unable to determine from a process design in BPMN what the effects achieved by a process might be at any point in the process design.

This is the problem that this paper seeks to address. The problem is not alleviated by taking recourse to the formal semantics of process design notations such as BPMN. Such semantics, as pointed out above, only describe the coordination aspects of a process. In addition, there is no consensus on the semantics of BPMN [6].

The solution we propose involves explicit semantic annotation of process models by analysts. The problem is challenging for a variety of reasons. Analysts need to be provided with a simple and accessible mechanism to describe the effects of process steps. The language in which these effects need to be specified should ideally be formal, permitting sophisticated tool support for several of the analysis and reasoning tasks mentioned above. A formal language would however not be practitioner-accessible. Informal annotations, on the other hand, make substantive tool support for these analysis tasks difficult. The use of controlled natural language (CNL) [7] is an effective compromise between these two extremes, by offering the analyst a repertoire of sentence schemas in which to describe the effects - populating a sentence schema generates a correspondingly instantiated formal annotation.

To ensure practitioner accessibility, and to avoid placing an unduly heavy burden of annotation on analysts, our approach only requires that analysts provide a description of the *immediate effects* of each process task, i.e., a context-independent specification of the functionality (together with relevant associated ramifications) of each task. These are then accumulated into *cumulative effect annotations* in a context-sensitive manner, such that the cumulative effect annotations associated with any task in a BPMN process model would describe the effects achieved by the process were it to execute up to that point. We note that such a description will necessarily be non-deterministic, i.e., there might be alternative *effect scenarios* that might transpire if a process has executed up to a certain point in a process model. The non-determinism stems from two sources. First, a process might have taken different paths through a process model to arrive at a certain point. Second, the effects of certain process steps might "undo" the effects of prior process steps. This is often described as the *belief update* or *knowledge update* problem - multiple alternative means of resolving the inconsistencies generated by the "undoing" of effects is another source of non-determinism.

This paper describes a conceptual underpinning and implementation of the ProcessSEER tool (implemented using the Eclipse environment, the STP BPMN modelling tool [8], the Prover9 theorem prover [9] and the ACE-CNL controlled natural language toolkit [7]). In section 2 we discuss the background and relate our work to other current research in this field. Section 3 is broken up into a number of subsections that explore how effect annotations are accumulated across process models. It includes a procedure for accumulating *scenario labels* that effectively describe the paths taken through a process model to obtain the corresponding *effect scenarios*. In section 4 we describe how this machinery might leverage ontology, CNL, and semantic web services to obtain richer effect descriptions. Section 5 describes the design and implementation of a tool (Fig.1) that realises these functionalities.

2. Background and Related Work

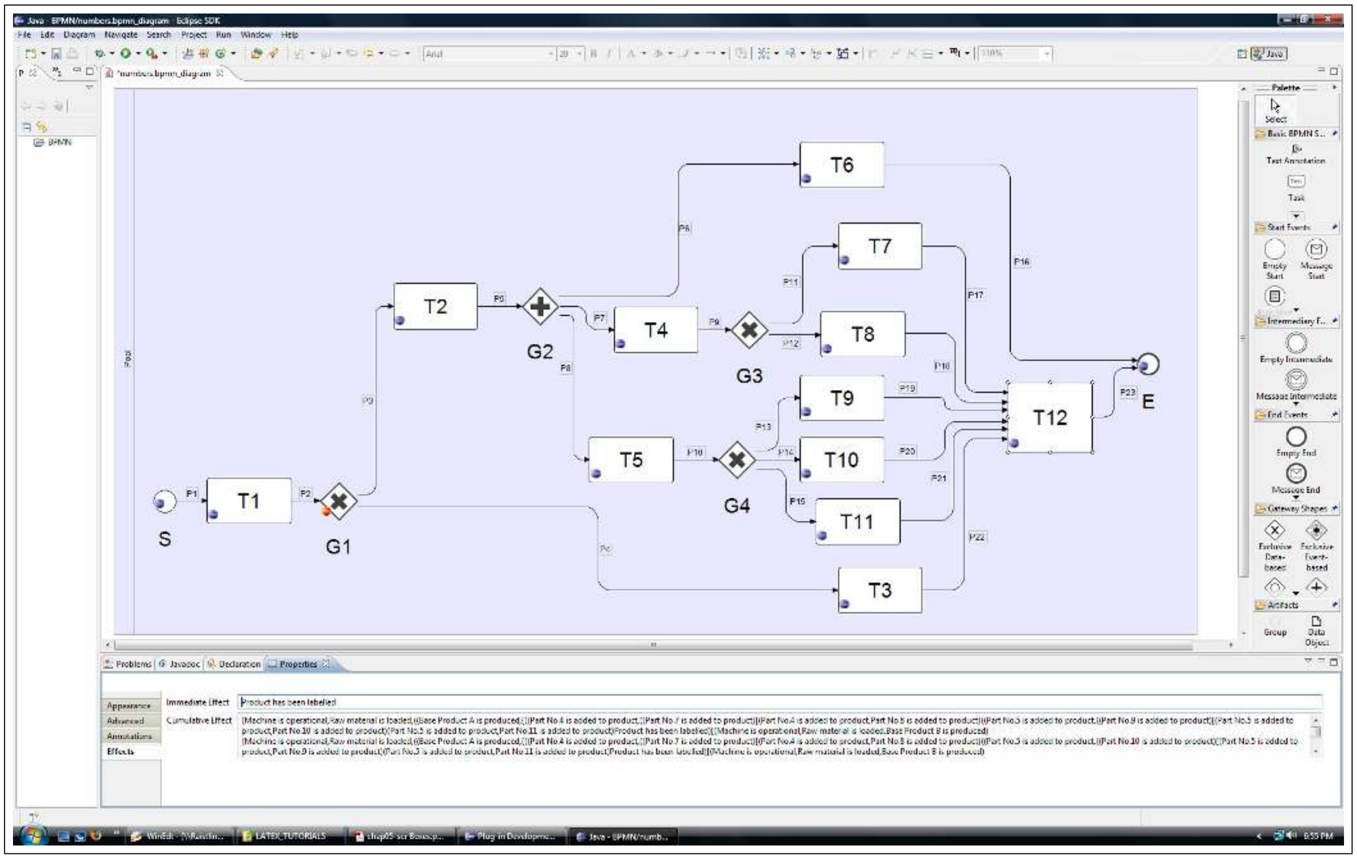
Annotating and analysing specifications of program functionality, in order to help establish program correctness, has a long tradition dating back to the introduction of the axiomatic techniques proposed by Hoare and Dijkstra [10]. With sufficient information, these forms of annotations provide [11] a basis for answering questions relating the identification of: the conditions enabling a process to be performed (i.e. postdiction); the conditions resulting from a process being performed in some context (i.e. prediction); and, the processes with the capability of realising a set of conditions when executed in some context (i.e. planning). Recently, similar proposals have emerged in the domain of web services [12] [13]. These forms of specification can be effective for performing analyst related tasks, however their utility and availability in some situations can be limited (e.g. cost restrictions) - warranting a need for "partiality" and "lightweight" approaches [14]. The contribution in this paper are techniques to leverage a partial specification of functional effects annotated to business process models.

A lot of effort is currently being directed into semantic annotation for web service or process discovery. Recently, a semantic annotation framework was developed to facilitate the interchange of process models and their discovery [15]. Ontology is used in this framework as a classification repository for the identification of processes or subprocesses that satisfy the selection criteria. Our tool will reduce the risk of modifying existing processes by alerting the analyst to the consequences of design time decisions. We use ontology in conjunction with a CNL taxonomy to define the vocabulary used in the effect annotations for the purpose of translation into formal logic. Our process differs from that described in [15] in that effect annotations are not simply used for term comparison but also for reasoning about process outcomes.

In [16], a Generic Process Model (GPM) is proposed to encode and extend the representation of processes with state and stability (i.e. goal) relevant information. These notions of state and stability lead to a general notion of validity of process models (primarily w.r.t. goal reachability). In [17], the GPM is used as a basis for identifying the scope of changes that can be made to an existing process given changes to GPM-related phenomena (e.g. goal change). Some of the techniques outlined in this paper, such as the accumulation procedure, help leverage partial and symbolic state descriptions to perform goal and change relevant analysis. In the SBPV approach [18], a scheme for annotating and propagating a restricted form of axiomatic task descriptions is introduced for a restricted class of process models, but differs in several key ways to our work. Our approach provides a parsimonious extension to the modelling framework (the analyst's effort is only extended by requiring immediate effect specifications of tasks in the BPMN model) and is driven by the need to identify the minimal amount of semantic annotation required to meet the requirements of functions such as compliance management, process change and lifecycle management, enterprise process architectures etc. The SBPV approach, on the other hand, requires complete specifications of both pre-conditions and post-conditions that are context-sensitive, thus placing a somewhat onerous burden on the analyst (besides additional annotations required for reachability analysis, which we do not consider). Our machinery for contextualising context-independent task effect specifications provided by analysts solves a harder problem, by permitting non-determinism in effect scenarios. We consequently cannot provide polynomial-time guarantees as the SBPV framework can. We believe this is not a significant impediment since design, annotation and propagation tasks do not normally involve real-time constraints, and afford the luxury of slower off-line computation. As our evaluation shows, we still are able to meet reasonable processing-time bounds. In [19], similar process annotation techniques are used for compliance checking.

3. Accumulating Effects

Our objective is to devise tool support that enables analysts to associate immediate (context independent) effects with process steps, so that the tool is then able to contextualise these effects, i.e., compute cumulative effects. Ultimately, we need a tool that answers the following question about any step in a process design: "What would the process have done if it had executed up to this point?". The answer to this question is non-deterministic and is provided in terms of



Immediate Effect: Product has been labelled

Cumulative Effect: [Machine is operational,Raw material is loaded,({Base Product A is produced,{[Part No.4 is added to product,{[Part No.7 is added to product]|{Part No.4 is added to product,Part No.6 is added to product]|{Part No.5 is added to product,({Part No.9 is added to product]{[Part No.5 is added to product,Part No.10 is added to product]{Part No.5 is added to product,Part No.11 is added to product]Product has been labelled]{[Machine is operational,Raw material is loaded,Base Product B is produced]
[Machine is operational,Raw material is loaded,({Base Product A is produced,{[Part No.4 is added to product,{[Part No.7 is added to product]|{Part No.4 is added to product,Part No.8 is added to product]|{Part No.5 is added to product,({Part No.10 is added to product]{[Part No.5 is added to product,Part No.9 is added to product]{Part No.5 is added to product,Part No.11 is added to product]Product has been labelled]|{Machine is operational,Raw material is loaded,Base Product B is produced]

Figure 1. The BPMN modelling tool with effect annotations displayed at the bottom.

(possibly multiple) *effect scenarios*. An effect scenario at a given point in a process is one consistent set of (cumulative) effects of a process if it were to execute up to that point. There are two reasons why we might find multiple effect scenarios associated with a given point in a process. First, a process might arrive at a given point through multiple paths, which cannot be predicted at design time. Second, activities in a process might *undo* the effects of activities earlier in the process (as we shall see in an example later in this section).

ProcessSEER performs on-demand, anytime computation of cumulative effects. There are two stages to effect accumulation. The first stage in effect accumulation involves deriving a *scenario label* [20] which provides the organising locus for our procedure. For obtaining the effect scenario at a given point in a process we compute the set of scenario labels at that point. A scenario label is a precise list of tasks that define a path leading from the Start Event in a model to the selected task. The simplest form of scenario label is a sequence of tasks. For example, in (Fig.1), if the task T_6 was selected, then a scenario label associated with that task would be $\langle S, T_1, G_1, T_2, G_2, T_6 \rangle$ where S is the start event. A scenario label can either be a sequence, denoted by the $\langle\rangle$ delimiters, or a set denoted by the $\{\}$ delimiters or combinations of both. The set delimiters are used to deal with parallel splits, and distinct elements in a set can be performed in any order.

The second stage of effect accumulation involves the processing of immediate effect annotations for each of the tasks listed in the scenario label using a pair-wise operation where the immediate effect of S is combined with the immediate effect of T_1, the result being the cumulative effect at T_1. The cumulative effect at T_1 is then combined with the immediate effect of T_2 resulting in the cumulative effect at T_2 and so on up to T_n.

3.1. A Procedure for Effect Accumulation over Scenario Labels

Contiguous Tasks: We define a process for *pair-wise effect accumulation*, which, given an ordered pair of tasks with effect annotations, determines the cumulative effect after both tasks have been executed in contiguous sequence. We assume throughout, the existence of a background knowledge-base (KB) that provides an additional basis for consistency. Consider the following simple example, where task T_2 follows task T_1, such that T_2 somehow "undoes" the effects of T_1 or changes the status of some entity referred to in T_1. For instance, the status of a cheque submitted in T_1 might be "not yet cleared", while the immediate effect of the "cheque clearance" task T_2 might be to set its status to "cleared". A background rule that specifies that a cheque cannot have a "cleared" and "not yet cleared" status simultaneously ensures that we do not counter-intuitively obtain both status descriptions in the same effect scenario.

The procedure serves as a methodology for analysts to follow if only informal annotations are available. We assume that the effect annotations have been represented in conjunctive normal form (CNF) where each clause is also a *prime implicate* [21] (this provides a non-redundant canonical form). Simple techniques exist for translating arbitrary sentences into the conjunctive normal form, and for obtaining the prime implicates of a theory (references omitted for brevity). Let $\langle T_i, T_j \rangle$ be an ordered pair of tasks connected via a sequence flow such that T_i precedes T_j, let e_i be an effect scenario associated with T_i and e_j be the immediate effect annotation associated with T_j. Let $e_i = \{c_{i1}, c_{i2}, \ldots, c_{im}\}$ and $e_j = \{c_{j1}, c_{j2}, \ldots, c_{jn}\}$ (we can view CNF sentences as sets of clauses, without loss of generality). If $e_i \cup e_j$ is consistent, then the resulting cumulative effect, denoted by $acc(e_i, e_j)$, is $e_i \cup e_j$. Else, we define $e'_i \subseteq e_i$ such that $e'_i \cup e_j$ is consistent and there exists no e''_i such that $e'_i \subset e''_i \subseteq e_i$ and $e''_i \cup e_j$ is consistent. We define $acc(e_i, e_j) = e'_i \cup e_j$. We note that $acc(e_i, e_j)$ is non-unique i.e. there are multiple alternative sets that satisfy the requirements for e_i. In other words, the cumulative effect of the two tasks consists of the effects of the second task plus as many of the effects of the first task as can be consistently included. We remove those clauses in the effect annotation of the first task that contradict the effects of the second task. The remaining clauses are undone, i.e., these effects are overridden by the second task.

In the preceding, we assume that all consistency checks implicitly include a background knowledge base (KB) containing rules and axioms. Thus, the statement that $e'_i \cup e_j$ is consistent, effectively entails $e'_i \cup e_j \cup KB$ is consistent. We omit references to KB for ease of exposition. The following example illustrates an application of this definition.

Example: Let e_1 and e_2 represent effect annotations at T_1 and T_2 in a process model where T_2 immediately follows T_1. Let e_1 represent a cumulative effect annotation, i.e., an effect scenario, while e_2 represents an immediate effect annotation. At T_1 the cumulative effect is $(p \wedge q)$ and the immediate effect of T_2 is r. A rule exists in the KB that states $KB = r \rightarrow \neg(p \wedge q)$.

$e_1 = (p \wedge q)$
$e_2 = r$
$KB = r \rightarrow \neg(p \wedge q)$
$\neg(p \wedge q) \equiv (\neg p \vee \neg q)$

Applying the definition above, the two alternative effect scenarios describing the cumulative effects at T_2 are $\{p, r\}$ and $\{q, r\}$.

In addition to pair-wise effect accumulation across scenario labels, we need to make special provision for the following: (1) accumulation across AND-joins, and (2) accumulation of effects over message flows (extending the framework presented in [20]). Consider the scenario label $\langle S, T_h, \{\langle T_{i1}, T_{i2}, \ldots, T_{in}\rangle, \langle T_{j1}, T_{j2}, \ldots, T_{jm}\rangle\}, T_k\rangle$. Let the immediate effects of T_{i1}, T_{j1} and T_k be e_{i1}, e_{j1} and e_k respectively. Let E_h, E_{in} and E_{jm} be the set of cumulative effect scenarios associated with T_h, T_{in} and T_{jm} respectively. The set of cumulative effect scenarios associated with T_{i1} is given by $\{acc(e, e_{i1}) \mid e \in E_h\}$. Similarly, the set of cumulative effect scenarios associated with T_{j1} is given by $\{acc(e, e_{j1}) \mid e \in E_h\}$. In other words, we accumulate over the pair of tasks $\langle T_h, T_{i1}\rangle$ as if they constitute a contiguous pair (and similarly for the pair $\langle T_h, T_{j1}\rangle$). We accumulate across AND-joins in the following manner. The set of cumulative effect scenarios associated with T_h is given by $\{acc(es_i, e_k) \cup acc(es_j, e_k) \mid es_i \in E_{in}, es_j \in E_{jm}$ and es_i, es_j are exclusion-compatible$\}$. In other words, we pair-wise accumulate the immediate effect of T_k with each effect scenario of each of tasks preceding the AND-join, but then combine them via set union since every possible combination of the prior scenarios could potentially transpire. *Exclusion-compatibility* provides a guarantee that the effect scenarios could potentially occur together, i.e., that they do not have a mutually exclusive (XOR) split in their antecedents relative to each other. Exclusion-compatibility is determined using the *exclude-set* mechanism described in the next section. Note that we do not consider the possibility of a pair of effect scenarios es_{1i} and es_{2j} being inconsistent, since this would only happen in the case of intrinsically and obviously erroneously constructed process models.

Much of the earlier and following discussion pertains to flows within individual pools. Message flow links across pools can be dealt with in a relatively straightforward fashion by requiring an immediate effect annotation for each incoming message. These effects are combined via conjunction with the immediate effects of the task associated with the incoming message. We assume again that no inconsistencies appear between the message and task effects - such inconsistencies would only appear in erroneous process designs.

The procedure described above does not satisfactorily deal with loops, but we can perform approximate checking by partial loop unravelling e.g., assume that the loop is executed n times where n is set by the analyst. Analysts can also identify infeasible (with respect to domain constraints) effect scenarios obtained in the process. We note that our objective is to devise decision-support functionality in the compliance management space, with human analysts vetting key changes before they are deployed.

3.2. A Procedure for Computing Scenario Labels

As previously mentioned the process of effect accumulation is reliant upon the computation of scenario labels that act like a road map for governing the order in which effect annotations are assembled and analysed. The computation of scenario labels at a given point is a non-trivial exercise. Of particular concern is the fact that scenario labels require selective sorting and complex processing to identify *exclude sets*, i.e., sequences that are explicitly excluded from effect accumulation.

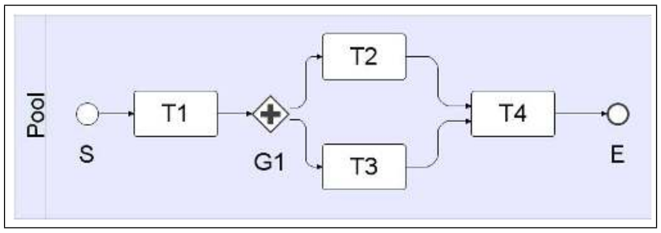


Figure 2. A BPMN model containing a parallel gateway split and an informal join.

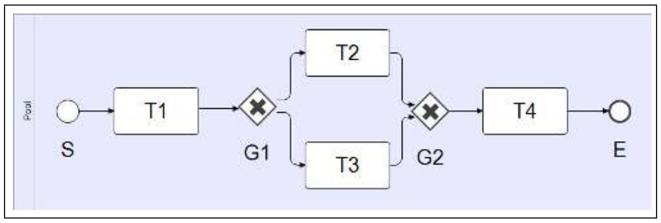


Figure 3. A BPMN model containing an exclusive gateway split and join.

Selective sorting: There is a unique sorting requirement for scenario labels. A scenario label is not necessarily a sequence, yet it can contain sequential elements. The scenario label $\langle\langle S, T_1, G_1, \{\langle T_2\rangle, \langle T_3\rangle\}, G_2, T_4\rangle, [\emptyset]\rangle$ from (Fig.2) shows how tasks T_2 and T_3 are both independent subsequences contained within a set, indicating that the separate subsequences can occur in any order. The set of subsequences is regarded as a separate element within an outer sequence of tasks. Although the execution of subsequences within a set may occur in any order, the set must occur in sequential order with other elements in the scenario label. Paths can be traced and stitched together in any order. Tasks contain no priority information so sorting must occur during the scenario label assembly process.

Complex processing and exclude sets: In (Fig.3) there are two scenario labels generated when T_4 is selected:

Label 1: $\langle\langle S, T_1, G_1, T_2, G_2, T_4\rangle, [\langle S, T_1, G_1, T_3\rangle]\rangle$

Label 2: $\langle\langle S, T_1, G_1, T_3, G_2, T_4\rangle, [\langle S, T_1, G_1, T_2\rangle]\rangle$

Each scenario label indicates a possible list of tasks that could have occurred in order to arrive at task T_4. Each scenario label also contains an 'exclude set' (denoted by "[ ]") that explicitly defines paths that may not be traversed given the current path. In scenario label 1 the sequence of tasks (path) followed to arrive at T_4 is $\langle S, T_1, G_1, T_2, G_2, T_4\rangle$ and the path to be explicitly excluded is the sequence $\langle S, T_1, G_1, T_3\rangle$. The exclude set is meant to assist the reasoning engine in detecting inconsistencies like exclusive gateway splits being accidently merged using parallel gateway joins.

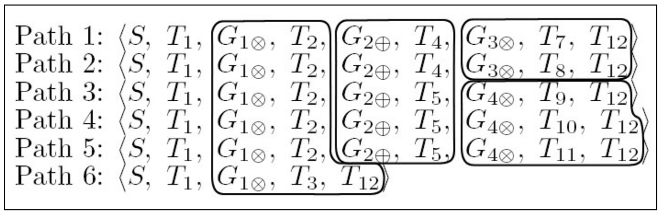
3.2.1. Scenario Label Extraction. The Scenario Label Extraction Process is divided into three steps. The first step involves a reverse path traversal method where the selected activity becomes the root node in a tree structure and all branches are traced back to the start event. The second step segments the list of paths into a set of gateways and their subsequences. These segments are then used in the final step to produce a list of scenario labels.

Reverse path traversal: All elements in the BPMN model are treated as generic nodes in a tree structure and the Path Collection Procedure walks the model in a reverse depth-first search from the selected object back to the Start Event. This is a recursive function that returns a list of Paths, i.e., a list of sequences of tasks and gateways. Each Path discovered is completely sequential. It does not contain any parallel occurrences because all gateway influences are ignored.

Using the Path Collection Procedure with (Fig.1) and selecting task T_{12} produces the following list of paths:

Path List:
Path 1: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_4,\ G_{3\otimes},\ T_7,\ T_{12}\rangle$
Path 2: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_4,\ G_{3\otimes},\ T_8,\ T_{12}\rangle$
Path 3: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_5,\ G_{4\otimes},\ T_9,\ T_{12}\rangle$
Path 4: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_5,\ G_{4\otimes},\ T_{10},\ T_{12}\rangle$
Path 5: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_5,\ G_{4\otimes},\ T_{11},\ T_{12}\rangle$
Path 6: $\langle S,\ T_1,\ G_{1\otimes},\ T_3,\ T_{12}\rangle$

The symbols $\otimes$ and $\oplus$ are used to denote exclusive and parallel gateways respectively. This list represents all possible routes from the start event S to task T_{12}. The procedure eliminates extraneous parallel pathways such as $\langle S, T_1, G_{1\otimes}, T_2, G_{2\oplus}, T_6\rangle$ by tracing backward from T_{12}. This list of paths is then segmented into collections of subsequences that we call *gateway sequences*.



Path 1: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_4,\ G_{3\otimes},\ T_7,\ T_{12}\rangle$
Path 2: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_4,\ G_{3\otimes},\ T_8,\ T_{12}\rangle$
Path 3: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_5,\ G_{4\otimes},\ T_9,\ T_{12}\rangle$
Path 4: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_5,\ G_{4\otimes},\ T_{10},\ T_{12}\rangle$
Path 5: $\langle S,\ T_1,\ G_{1\otimes},\ T_2,\ G_{2\oplus},\ T_5,\ G_{4\otimes},\ T_{11},\ T_{12}\rangle$
Path 6: $\langle S,\ T_1,\ G_{1\otimes},\ T_3,\ T_{12}\rangle$

Figure 4. Gateway Sequence Groupings.

Extracting gateway sequences: The segmentation of the path list in (Fig.4) identifies the data captured in gateway sequences. The boxed diagrams in (Fig.5) represent instances of the gateway sequence class and their contents. Each gateway sequence contains the gateway and its downstream subsequences. The subsequences either end at the selected activity or at another gateway. If a subsequence ends with a gateway then that gateway is itself a gateway sequence. Therefore gateway sequence G_2 is inserted at the end of the first subsequence in gateway sequence G_1. Likewise gateway sequences G_3 and G_4 are both inserted into gateway sequence G_2. The first step in extracting gateway sequences

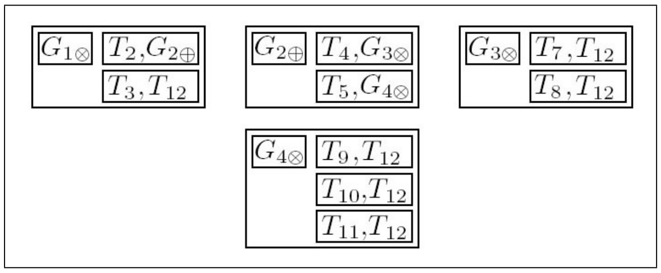


Figure 5. Gateway Sequence Objects.

from the path list is to find all the gateways. There are no gateway joins used in (Fig.1) but if there were then they must be filtered out of the gateway collection process because they have no effect on the subsequences or sets in a scenario label. All we are interested in at this stage is collecting gateway splits. Gateway joins are detected and filtered out by identifying multiple incoming edges and only a single outgoing edge. The Gateway Split Collection Procedure is a simple iterative process that doesn't require much explanation other than that a gateway identification method is employed to first determine whether the object is a gateway and second, whether it is a gateway split or a gateway join.

Extracting all gateway splits from the path list produces the following list:

$$G_{1\otimes},\ G_{2\oplus},\ G_{3\otimes},\ G_{4\otimes}$$

Iterating forward along each path tends to collect gateways in the order in which they appear in the BPMN diagram. The order is somewhat important when building gateway sequences because child gateway sequences need to be inserted into parents. Therefore children must already exist before their parents can be created. Reverse iteration through the gateway list is employed to capture children first. It cannot be guaranteed that all gateway splits in the gateway list will be in order so numerous passes through the list may be necessary. In a case where the gateway list is assembled out-of-order, a subprocedure checks for the existence of child gateway sequences and aborts parent creation if they are missing. If the gateway is successfully processed then it is removed from the gateway list and the list is recycled to collect gateways that were missed. An ordered collection of Gateways is therefore purely an efficiency enhancing operation.

Gateway sequences create a new model that is independent of the original list of paths. This facilitates the generation of a completely independent list of scenario labels. Each gateway sequence object contains a method for assembling its subsequences with its parent's subsequence. This method is the final step in the overall Scenario Label Extraction Process and is covered in the next section.

Deriving scenario labels: The first gateway sequence is all that is necessary to derive all scenario labels. This is because each gateway sequence is responsible for merging itself with its parent's subsequence, i.e., the parent gateway sequence passes its subsequence to the child gateway sequence and the child builds the appropriate number of subsequences by combining the parent subsequence with its own subsequences, returning the appended list of subsequences back to the parent. Deriving all scenario labels begins by creating a list of the first tasks in the model that are common to all paths. The common tasks at the beginning of every path in the path list are $\langle S,\ T_1\rangle$. That sequence is then passed to $G_{1\otimes}$, the first gateway sequence in the list. That parent gateway sequence then passes its subsequences to its children which in turn return their own sets of subsequences. The process is explained in the following sequence of operations. The symbol $\rightarrow$ is used to indicate that data is passed to a child gateway sequence while the symbol $\leftarrow$ is used to indicate that data is passed back to the parent gateway sequence.

The beginning subsequence common to all Paths is passed to the first gateway sequence.

$$\langle S,\ T_1\rangle \rightarrow\ G_{1\otimes}$$

Gateway sequence $G_{1\otimes}$ must first process its child gateway sequences before it returns its final list.

$$\langle S,\ T_1,\ G_{1\otimes}, \langle T_2,\ G_{2\oplus}\rangle\rangle$$
$$\langle S,\ T_1,\ G_{1\otimes}, \langle T_3,\ T_{12}\rangle\rangle$$

$G_{1\otimes}$ passes its subsequence to its child gateway sequence.

$$\langle T_2\rangle \rightarrow\ G_{2\oplus}$$

$G_{2\oplus}$ must first process its child gateway sequences before it returns its final list. Note the difference in sequence assembly between exclusive $G_{1\otimes}$ and parallel $G_{2\oplus}$.

$$\langle T_2,\ G_{2\oplus}\{\langle T_4,\ G_{3\otimes}\rangle, \langle T_5,\ G_{4\otimes}\rangle\}\rangle$$

$G_{2\oplus}$ passes each of its subsequences to each of its child gateway sequences $G_{3\otimes}$ and $G_{4\otimes}$.

$$\langle T_4\rangle \rightarrow\ G_{3\otimes}$$
$$\langle T_5\rangle \rightarrow\ G_{4\otimes}$$

$G_{3\otimes}$ and $G_{4\otimes}$ assemble their subsequences with the parent subsequence and return their list of subsequences back to the parent $G_{2\oplus}$.

$$G_{2\oplus}\ \leftarrow \langle T_4,\ G_{3\otimes},\ T_7,\ T_{12}\rangle$$

$G_{2\oplus} \leftarrow \langle T_4,\ G_{3\otimes},\ T_8,\ T_{12}\rangle$
$G_{2\oplus} \leftarrow \langle T_5,\ G_{4\otimes},\ T_9,\ T_{12}\rangle$
$G_{2\oplus} \leftarrow \langle T_5,\ G_{4\otimes},\ T_{10},\ T_{12}\rangle$
$G_{2\oplus} \leftarrow \langle T_5,\ G_{4\otimes},\ T_{11},\ T_{12}\rangle$

$G_{2\oplus}$ appends its parent subsequence to each of the returned child subsequences and returns that list to $G_{1\otimes}$. Note how the parallel gateway sequence assembles each subsequence returned from $G_{3\otimes}$ with each of the subsequences returned from $G_{4\otimes}$. $G_{2\oplus}$ now returns six subsequences to $G_{1\otimes}$ from the original five subsequences it received from its child gateways $G_{3\otimes}$ and $G_{4\otimes}$.

$G_{1\otimes} \leftarrow \langle T_2, G_{2\oplus}\{\langle T_4, G_{3\otimes}, T_7, T_{12}\rangle, \langle T_5, G_{4\otimes}, T_9, T_{12}\rangle\}$
$G_{1\otimes} \leftarrow \langle T_2, G_{2\oplus}\{\langle T_4, G_{3\otimes}, T_7, T_{12}\rangle, \langle T_5, G_{4\otimes}, T_{10}, T_{12}\rangle\}$
$G_{1\otimes} \leftarrow \langle T_2, G_{2\oplus}\{\langle T_4, G_{3\otimes}, T_7, T_{12}\rangle, \langle T_5, G_{4\otimes}, T_{11}, T_{12}\rangle\}$
$G_{1\otimes} \leftarrow \langle T_2, G_{2\oplus}\{\langle T_4, G_{3\otimes}, T_8, T_{12}\rangle, \langle T_5, G_{4\otimes}, T_9, T_{12}\rangle\}$
$G_{1\otimes} \leftarrow \langle T_2, G_{2\oplus}\{\langle T_4, G_{3\otimes}, T_8, T_{12}\rangle, \langle T_5, G_{4\otimes}, T_{10}, T_{12}\rangle\}$
$G_{1\otimes} \leftarrow \langle T_2, G_{2\oplus}\{\langle T_4, G_{3\otimes}, T_8, T_{12}\rangle, \langle T_5, G_{4\otimes}, T_{11}, T_{12}\rangle\}$

$G_{1\otimes}$ now appends its own subsequence, including the start sequence it received, to each of the child subsequences returned by $G_{2\oplus}$.

$\langle S, T_1, G_1, \langle T_2, G_2\{\langle T_4, G_3, T_7, T_{12}\rangle, \langle T_5, G_4, T_9, T_{12}\rangle\}$
$\langle S, T_1, G_1, \langle T_2, G_2\{\langle T_4, G_3, T_7, T_{12}\rangle, \langle T_5, G_4, T_{10}, T_{12}\rangle\}$
$\langle S, T_1, G_1, \langle T_2, G_2\{\langle T_4, G_3, T_7, T_{12}\rangle, \langle T_5, G_4, T_{11}, T_{12}\rangle\}$
$\langle S, T_1, G_1, \langle T_2, G_2\{\langle T_4, G_3, T_8, T_{12}\rangle, \langle T_5, G_4, T_9, T_{12}\rangle\}$
$\langle S, T_1, G_1, \langle T_2, G_2\{\langle T_4, G_3, T_8, T_{12}\rangle, \langle T_5, G_4, T_{10}, T_{12}\rangle\}$
$\langle S, T_1, G_1, \langle T_2, G_2\{\langle T_4, G_3, T_8, T_{12}\rangle, \langle T_5, G_4, T_{11}, T_{12}\rangle\}$
$\langle S, T_1, G_1, \langle T_3, T_{12}\rangle\}$

The result is that a complete list of scenario labels is returned from gateway sequence $G_{1\otimes}$. For the purpose of simplicity exclude sets have been omitted from the list. However, exclude sets are calculated by exclusive gateway sequences and returned as part of the assembly process. An exclusive gateway sequence contains a complete list of all its subsequences and excludes every other subsequence from each of its subsequences.

The final list of scenario labels including exclude sets can be seen in (Fig.6). Each scenario label contains a list of elements that can be tasks, gateways, sequences, exclude sets and parallel sets. Each element in a scenario label can be identified as one of these objects so that task grouping rules can be applied. Once the scenario labels have been assembled then immediate effect annotations can be easily extracted from the base elements (tasks) and interpreted according to their placement within the scenario labels. The process of extracting the immediate effect annotations and accumulating the effects is explained in more detail in section 3.1.

Results: To avoid confusion the output in (Fig.6) specifically uses square brackets "[ ]" to represent exclude sets and braces "{ }" to represent parallel sets. When task T_{12} is selected in (Fig.1) and these procedures applied, they generate the set of scenario labels listed in (Fig.6). This demonstrates the tool's ability to process complex BPMN models. It is worth noting that the number of scenario labels generated from (Fig.1), seven in total, is greater than the six paths listed in the path list. The effect of gateways is not considered when tracing paths. However, when deriving scenario labels, the gateway effect is considered. Parallel gateways will cause multiple paths to collapse into a single scenario label that includes parallel sets whereas exclusive gateways, when nested inside parallel gateways will generate a greater number of scenario labels to accommodate for all possible combinations. For example, a parallel split followed by two exclusive splits, each containing three alternate paths, will generate six paths. However, because each exclusive path from one split must be combined with each exclusive path from the other there will be $3 \times 3 = 9$ scenario labels generated. This illustrates how scenario labels are quantifiably independent from traceable paths.

4. Leveraging Ontology, Controlled Natural Language and Semantic Web Services

Ontology, a relationship reference model of domain specific terminology, can ease the semantic annotation exercise in several ways (as we have noted earlier, a substantial body of work explores the role of ontology in this setting). First, an ontology can help avoid *naming conflicts* in analyst-mediated immediate effect annotation (i.e., the same concept being referred to by different names). Second, an ontology can help resolve *abstraction conflicts*, where effect descriptions are provided at different levels of abstraction, and therefore in different vocabularies. Third, an ontology can provide the background knowledge base referred to earlier that is used for consistency checking of effect accumulation. Such a background KB can be easily obtained by extraction from an ontology those relations (rules) whose *concept signatures* are included in the concept signature of the effect annotations provided (i.e., that refer precisely to those concepts referred to in the effect annotations). Finally, we can envisage an extension to our current tool that leverages ontology to generate suggestions to analysts in terms of additional (immediate) effects that might be included in the annotation (for instance, a causal rule *a* **causes** *b* that might be implicit in an ontology might be used to suggest to an analyst that *b* be included in an annotation that contains *a*.

We use CNL (Controlled Natural Language) [22] as the notation in which an analyst specifies the immediate effects of process tasks. CNL provides a practitioner-accessible

1: $\langle S, T_1, G_1, \langle\langle T_2, G_2, \{\langle T_4, G_3, \langle\langle T_7\rangle, [\langle T_4, G_3, T_8\rangle]\rangle\rangle, \langle T_5, G_4, \langle\langle T_9\rangle, [\langle T_5, G_4, T_{10}\rangle, \langle T_5, G_4, T_{11}\rangle]\rangle\rangle\}, T_{12}\rangle, [\langle S, T_1, G_1, T_3\rangle]\rangle\rangle$

2: $\langle S, T_1, G_1, \langle\langle T_2, G_2, \{\langle T_4, G_3, \langle\langle T_7\rangle, [\langle T_4, G_3, T_8\rangle]\rangle\rangle, \langle T_5, G_4, \langle\langle T_{10}\rangle, [\langle T_5, G_4, T_9\rangle, \langle T_5, G_4, T_{11}\rangle]\rangle\rangle\}, T_{12}\rangle, [\langle S, T_1, G_1, T_3\rangle]\rangle\rangle$

3: $\langle S, T_1, G_1, \langle\langle T_2, G_2, \{\langle T_4, G_3, \langle\langle T_7\rangle, [\langle T_4, G_3, T_8\rangle]\rangle\rangle, \langle T_5, G_4, \langle\langle T_{11}\rangle, [\langle T_5, G_4, T_9\rangle, \langle T_5, G_4, T_{10}\rangle]\rangle\rangle\}, T_{12}\rangle, [\langle S, T_1, G_1, T_3\rangle]\rangle\rangle$

4: $\langle S, T_1, G_1, \langle\langle T_2, G_2, \{\langle T_4, G_3, \langle\langle T_8\rangle, [\langle T_4, G_3, T_7\rangle]\rangle\rangle, \langle T_5, G_4, \langle\langle T_9\rangle, [\langle T_5, G_4, T_{10}\rangle, \langle T_5, G_4, T_{11}\rangle]\rangle\rangle\}, T_{12}\rangle, [\langle S, T_1, G_1, T_3\rangle]\rangle\rangle$

5: $\langle S, T_1, G_1, \langle\langle T_2, G_2, \{\langle T_4, G_3, \langle\langle T_8\rangle, [\langle T_4, G_3, T_7\rangle]\rangle, \langle T_5, G_4, \langle\langle T_{10}\rangle, [\langle T_5, G_4, T_9\rangle, \langle T_5, G_4, T_{11}\rangle]\rangle\rangle\}, T_{12}\rangle, [\langle S, T_1, G_1, T_3\rangle]\rangle\rangle$

6: $\langle S, T_1, G_1, \langle\langle T_2, G_2, \{\langle T_4, G_3, \langle\langle T_8\rangle, [\langle T_4, G_3, T_7\rangle]\rangle\rangle, \langle T_5, G_4, \langle\langle T_{11}\rangle, [\langle T_5, G_4, T_9\rangle, \langle T_5, G_4, T_{10}\rangle]\rangle\rangle\}, T_{12}\rangle, [\langle S, T_1, G_1, T_3\rangle]\rangle\rangle$

7: $\langle S, T_1, G_1, \langle\langle T_3, T_{12}\rangle,\ [\langle S, T_1, G_1, T_2\rangle]\rangle\rangle$

Figure 6. A final list of scenario labels with exclude sets.

language for specifying effects (and thus, avoid the problems associated with insisting that analysts become proficient in a formal notation) while still making it relatively easy to translate these into an underlying formal representation.

A domain specific ontology also facilitates the translation of CNL into First Order Logic (FOL). Languages such as OWL-S use variable names in their effect descriptions, e.g. *objectPurchased*. Such a term would not ordinarily exist within the taxonomy of a CNL application. A CNL interpreter (an application that can translate CNL into FOL) would require a domain specific ontology that includes such terms. Consider the following immediate effect annotation written in natural language:

> *The purchase amount is confirmed with a confirmation number and the client is the owner of the object purchased and the creditLimit of the credit card is decreased by the purchase amount.*

A CNL interpreter linked to an ontology that included business transaction related terminology could identify compound terms like *purchase amount* and suggest alternatives such as *purchaseAmount*. This is reliant upon the standardisation of terms within the domain specific ontology. A CNL that is becoming popular within the semantic web community is Attempto Controlled English (ACE) [23] [24]. The following sentences are written in ACE and represent a conversion of the immediate effect annotations above. They contain variable names but can be easily understood by an analyst.

> *A purchase has a purchaseAmount that is confirmed by a confirmationNumber.*
> *A purchase has an objectPurchased that is owned by a client.*
> *A creditCard has a creditLimit that is decreased by a purchaseAmount.*

From these CNL sentences the following paraphrases can be derived:

There is a purchase X1.
There is a confirmationNumber X2.
The confirmationNumber X2 confirms a purchaseAmount X3.
The purchase X1 has the purchaseAmount X3.

There is a client X4.
The client X4 owns an objectPurchased X5.
The purchase X1 has the objectPurchased X5.

There is a creditCard X6.
The purchaseAmount X3 decreases a creditLimit X7.
The creditCard X6 has the creditLimit X7.

The paraphrasing then allows us to assemble the following FOL statements that are machine processable:

has(purchase, purchaseAmount)
confirms(confirmationNumber, purchaseAmount)
owns(client, objectPurchased)
has(purchase, objectPurchased)
decreases(purchaseAmount, creditLimit)
has(creditCard, creditLimit)

Consider the background rule:

$\forall x, y, z(owns(x, z) \wedge \neg equals(x, y) \rightarrow \neg owns(y, z))$

indicating that two entities may not own a purchased object at the same time. Now suppose that the previous activity in a business process contained, as part of its cumulative effect, $owns(company, objectPurchased)$. This would be the case prior to the credit card transaction and confirmation of purchase. Consider now that the current task is to finalise the purchase and the immediate effect of that task is $owns(client, objectPurchased)$. When this annotation is combined with the previous cumulative effect and presented to the reasoning engine, an inconsistency will be detected, i.e., both company and client cannot own the object purchased. The pair-wise accumulation application will then replace the statement $owns(company, objectPurchased)$ with the updated statement $owns(client, objectPurchased)$ to produce the cumulative effect at the current task.

The OWL-S segment below [25] describes the effect of a purchase being confirmed for an agreed amount, and relates directly to the CNL and FOL specifications presented earlier in this section.

$< process : hasEffect >$
$< expr : KIF - Condition >$
$< expr : expressionBody >$
$(and\ (confirmed\ (purchase\ ?purchaseAmount)$
$?confirmationNumber)$
$(own\ ?objectPurchased)$
$(decrease\ (creditLimit\ ?creditCard)$
$?purchaseAmount))$
$< /expr : expressionBody >$
$< /expr : KIF - Condition >$
$< /process : hasEffect >$

Semantic web services can thus be directly leveraged to obtain immediate effect specifications of process tasks that these services might execute.

5. Implementation and evaluation

This ProcessSEER tool has been implemented using the Eclipse environment and the STP (SOA Tools Platform) BPMN modelling tool [8]. The tool supports analyst-mediated specification of immediate effects of process tasks within a process modelling environment. The tool then computes cumulative effect scenarios in an "on-demand" fashion (i.e., the tool computes the effect scenarios associated with tasks selected by the user). Pair-wise effect accumulation requires consistency checks - these are performed by the first-order theorem prover Prover9 [9]. First-order logic representations of controlled natural language effect specifications are obtained using the ACE-CNL toolkit [7]. The application interface in (Fig.1) shows the immediate and cumulative effects displayed in a dialogue box at the bottom of the screen (space and formatting restrictions prevented us from displaying a larger screen shot).

The ProcessSEER tool was evaluated in two ways. First, expert evaluation was conducted using experienced analysts with significant background in BPMN modelling, in the context of industry-scale process models (developed for the financial services sector) with approximately 20-22 activities each. The feedback was generally positive in relation to usability, effectiveness/accuracy of the effect scenarios generated and the response time. The testing has helped to improve tool performance and usability. The second component of the evaluation exercise involved testing for processing time with increasing size and complexity of models. A series of tests were conducted across a variety of models to evaluate the speed of scenario label derivation and pair-wise accumulation with consistency checking. These tests involved immediate effect specifications that translated into single first-order logic sentences for each activity. Each of these sentences were ground. This is a realistic assumption since quantified sentences rarely appear in the description of immediate effects, but more commonly appear in the rules in the background knowledge-base. The tests were conducted on a computer with a Pentium Quad Core 2.4GHz CPU and produced the results listed in Table 1

The results show the time required to compute an effect scenario for a selected activity, relative to the number of "upstream" activities to that activity, the number of paths that can be traced back to the start event from the selected activity and the number of scenario labels associated with the selected activity. Note that the number of distinct scenario labels need not equal the number of distinct paths, but might be greater or fewer. The recorded times represent the time taken for both the derivation of scenario labels and pair-wise accumulation of activities with consistency checking. The test data shows that dynamic accumulation can be realised within the operational constraints of the analyst.

Activities	Paths	Scenario Labels	milliseconds
10	12	4	209
12	6	7	248
20	8	8	515
22	13	48	2718
22	48	16	1376
24	13	45	2763
24	96	32	1519

Table 1. Effect accumulation speeds

Additional testing results are omitted here due to space restrictions, but the following observation is important. While the results above involve a significant number of consistency checks (one for every pair of contiguous activities), our evaluation suggests that this does not, in general, significantly degrade the performance of the tool. In the worst case, the computation of $acc(e_i, e_j)$ would require a number of consistency checks that is exponential in the cardinality of e_i. The theorem prover Prover9 takes approximately 3ms to check the consistency of a set of 50 sentences. Our evaluation suggests that the exponential number of consistency checks that might be required in the worst case does not significantly impact the processing time of the tool, for the following reasons. First, we believe inconsistent effects (where a task "undoes" the effects of a prior task) are relatively rare in business processes due to the contributory nature of activities toward achieving a final goal whereas they can be seen more frequently in, say, engineering processes. Second, the inconsistency at any given step typically involves a small number of effects (typically a single effect) of the most recent task. These can be directly identified by flagging the rules in the background knowledge base that are violated. The worst-case scenario involving an exponential number of consistency checks is therefore extremely rare. Third, several instances of inconsistency (such as those that involve resetting of status flags, e.g., from *not_paid* to *paid*) can be dealt with via specialised machinery that deals with inconsistent literals. Finally, the fact that the theorem prover is quite fast in the face of reasonable-sized sets of sentences provides further confidence that this will not be a major issue in practical settings.

6. Conclusion

A business process modelling tool that can provide instant feedback about the consequences of critical process re-engineering decisions at design time adds an extra layer of protection before changes are initiated. Add to this the ability to automatically translate business processes into semantic web services and we have a suite of tools of considerable benefit to any organisation.

ProcessSEER is currently capable of storing immediate effect annotations and dynamically accumulating those annotations with consistency checking for selected tasks in the model. The work involved in this paper represents a substantial contribution to the development of a BPMN modelling tool that can perform automated process analysis. Future development will extend the functionality, encouraging business contribution to semantic web content.

References

[1] A. Ghose and G. Koliadis, "Business process compliance: Techniques for design-time auditing and resolution," in *Proceedings of the First International Workshop on Juris-informatics (JURISIN07)*, 2007.

[2] G. Koliadis and A. Ghose, "Correlating business process and organizational models to manage change," in *Australasian Conference on Information Systems*, December 2006.

[3] C. Hall and P. Harmon, "The 2005 enterprise architecture, process modeling & simulation tools report," Tech. Rep., 2005.

[4] G. Koliadis, A. Vranesevic, M. Bhuiyan, A. Krishna, and A. Ghose, "A combined approach for supporting the business process model lifecycle," in *Proc. of the 10th Pacific Asia Conference on Information Systems (PACIS'06)*, 2006.

[5] OMG, *Business Process Modeling Notation (BPMN) Specification*, Object Management Group, February 2006. [Online]. Available: http://www.bpmn.org/Documents/OMG Final Adopted BPMN 1-0 Spec 06-02-01.pdf

[6] W. van der Aalst, "Pi calculus versus petri nets: Let us eat humble pie rather than further inflate the pi hype," *BPTrends*, vol. 3, no. 5, pp. 1–11, May 2004. [Online]. Available: http://is.tm.tue.nl/research/patterns/download/ bptrendsPiHype.pdf

[7] R. Schwitter and N. Fuchs, "Attempto - from specifications in controlled natural language towards executable specifications," *CoRR*, vol. cmp-lg/9603004, May 1996.

[8] "Eclipse soa tools platform, bpmn modeller," 2009. [Online]. Available: http://www.eclipse.org/bpmn/

[9] W. McCune, "Prover9," 2009. [Online]. Available: http://www.cs.unm.edu/ mccune/prover9/

[10] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[11] M. Shanahan, *Solving the Frame Problem - A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

[12] S. A. McIlraith, T. C. Son, and H. Zen, "Semantic web services," *IEEE Intelligent Systems*, vol. March/April, pp. 46–53, 2001.

[13] D. Martin and J. Domingue, "Semantic web services, part 2," *IEEE Intelligent Systems*, vol. November/December, pp. 8–15, 2007.

[14] D. Jackson and J. Wing, "Lightweight formal methods," *IEEE Computer*, pp. 21–22, 1996.

[15] Y. Lin, "Semantic annotation for process models: Facilitating process knowledge management via semantic interoperability," Ph.D. dissertation, Norwegian University of Science and Technology, Trondheim, Norway, March 2008.

[16] P. Soffer and Y. Wand, "Goal-driven analysis of process model validity," *Lecture Notes in Computer Science*, vol. 3084, pp. 521–535, June 2004.

[17] P. Soffer, "Scope analysis: identifying the impact of changes in business process models," *Software Process: Improvement and Practice*, vol. 10, no. 4, pp. 393–402, 2005.

[18] I. Weber, J. Hoffman, and J. Mendling, "Semantic business process validation," in *Proceedings of the 3rd International Workshop on Semantic Business Process Management*, 2008.

[19] G. Governatori, J. Hoffmann, S. Sadiq, and I. Weber, "Detecting regulatory compliance of business process models through semantic annotations," in *Proceedings of the 4th International Workshop on Business Process Design*, 2008.

[20] A. Ghose and G. Koliadis, "Pctk: A toolkit for managing business process compliance," in *Proceedings of the Second International Workshop on Juris-informatics (JURISIN'08)*, 2008.

[21] M. Raut and A. Singh, "Prime implicates of first order formulas," *International Journal of Computer Science and Applications*, vol. 1, 2004.

[22] R. Schwitter, "Home (controlled natural languages)," 2009. [Online]. Available: http://sites.google.com/site/controllednaturallanguage/

[23] N. E. Fuchs and R. Schwitter, "Web-annotations for humans and machines," in *Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*, ser. Lecture Notes in Computer Science. Springer, 2007.

[24] K. Kaljurand, "Attempto controlled english as a semantic web language," Ph.D. dissertation, Faculty of Mathematics and Computer Science, University of Tartu, 2007.

[25] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "Owl-s: Semantic markup for web services," Tech. Rep., November 2004. [Online]. Available: http://www.w3.org/Submission/OWL-S/