Goal-Driven Business Process Derivation*

Aditya K. Ghose[1], Nanjangud C. Narendra[2], Karthikeyan Ponnalagu[2], Anurag Panda[3], and Atul Gohad[3]

[1] University of Wollongong, Wollongong, Australia
{aditya.ghose}@gmail.com
[2] IBM Research India, Bangalore, India
{narendra,karthikeyan.ponnalagu}@in.ibm.com
[3] IBM India Software Lab, Bangalore, India
{panurag,agohad}@in.ibm.com

Abstract. Solutions to the problem of deriving business processes from goals are critical in addressing a variety of challenges facing the services and business process management community, and in particular, the challenge of quickly generating large numbers of effective process designs (often a bottleneck in industry-scale deployment of BPM). The problem is similar to the planning problem that has been extensively studied in the artificial intelligence (AI) community. However, the direct application of AI planning techniques places an onerous burden on the analyst, and has proven to be difficult in practice. We propose a practical yet rigorous (semi-automated) algorithm for business process derivation from goals. Our approach relies on being able to decompose process goals to a more refined collection of sub-goals whose ontology is aligned with that of the effects of available tasks which can be used to construct the business process. Once process goals are refined to this level, we are able to generate a process design using a procedure that leverages our earlier work on semantic effect annotation of process designs. We illustrate our ideas throughout this paper with a real-life running example, and also present a proof-of-concept prototype implementation.

Keywords: business process, goals, tasks, capabilities.

1 Introduction

One of the most crucial (and difficult) tasks in enterprises today is the derivation of business processes to meet stated business goals. Poor process derivation could result in wasteful and/or wrong tasks, and would require significant and costly rework to ensure that business process executions are able to adhere to their goals. Additionally, the requirement of business process compliance [1–3] - over and above the basic business goals - adds further complexity and difficulty.

Traditional approaches towards business process derivation from goals have focused on modeling this problem as an artificial intelligence (AI) planning problem [4], e.g., citations such as [5–7]. While such methods undoubtedly produce accurate solutions, they require specialist knowledge of planning and formal knowledge representation techniques that business analysts typically do not possess.

* Thanks to GR Gangadharan for his feedback.

Therefore, in this paper, we take a different approach. We assume the following inputs: (a) a set of process goals represented as a collection of boolean conditions in conjunctive normal form (CNF); (b) a capability library of existing tasks that can be used to satisfy the goals, with each annotated via its *effects* [8]; (c) a set of *domain constraints* that impose restrictions on how task execution in the business process should be sequenced. Given these inputs, the salient contribution of our paper is an algorithm for deriving a business process design from these inputs.

Our algorithm works as follows. First, the goals are successively refined using goal refinement strategies leveraged from the KAOS methodology [9, 10]. This refinement continues until there is an ontological match with the effects of the tasks in the capability library. Second, using the capability library, tasks are identified for each leaf-level goal. Third, precedence constraints among the tasks are derived from the given domain constraints. Finally, the business process design is generated from the goals and precedence constraints. Our approach also does not require the use of preconditions, which we show can be encoded via domain constraints.

2 Running Example

Our running example is a simplified version of an incident management process. Incidents are customer initiated calls based on service issues. The mission of incident management process is to handle all requests for problem solving and support in a consistent, timely and cost-effective manner. Typically, the process begins with a request from a client or with a problem statement highlighting the concerns of the client. It concludes with the client being satisfied with the response and the solution provided to solve the problem.

The goals and derived sub-goals of this process are depicted in Fig. 1. An AND link in Fig. 1 specifies that all sub-goals of a goal need to be satisfied for the goal to be satisfied; an XOR link specifies that the sub-goals are mutually exclusive, and only one is needed to satisfy the goal. For example, the goal of Incident and Problem management fulfills the goals Fix Problem, Detect Problem and Verify Problem, viz., a case of AND relationship. If we consider the goals Isolate Problem or Escalate Problem they share an XOR Relationship as in any given situation only one of the goals can be fulfilled and they are mutually exclusive in nature.

Some of the applicable domain constraints for this business process are: whether to escalate the problem to the next level, and whether a new incident should be linked to a previous incident in order to enhance reuse of earlier solutions.

3 Background

A *business process* is a sequence of *tasks*, with each task producing an *effect*. The accumulated effects of all task executions is the overall effect of the business process. The effect is the result of an activity executed by a cause or agent. Effects can be viewed as both: *normative* - as they state required outcomes (i.e., goals); and *descriptive* in that they describe the normal, and predicted, subset of all possible outcomes. We formally represent effect annotations using first-order logic. For simplicity, our business process

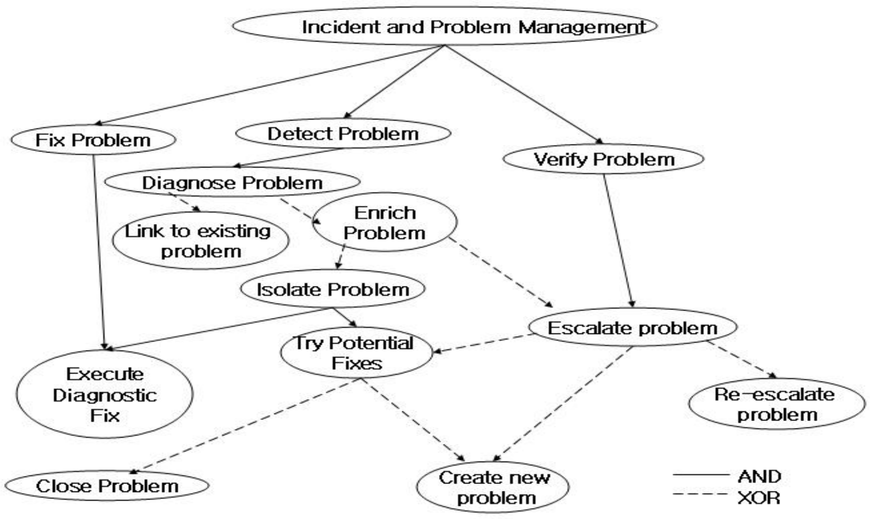


Fig. 1. Goals & Sub-Goals

does not contain loops; they can be incorporated into our business process model by abstracting them into single tasks.

We define a process for *pair-wise effect accumulation* [8, 11], which, given an ordered pair of tasks with effect annotations, determines the cumulative effect after both tasks have been executed in contiguous sequence. The procedure serves as an easily understandable yet rigorous methodology for analysts to follow. We assume that the effect annotations have been represented in conjunctive normal form (CNF) [12]. Simple techniques (e.g., [12]) exist for translating arbitrary sentences into CNF.

Let $< t_i, t_j >$ be the ordered pair of tasks, and let e_i and e_j be the corresponding pair of effect annotations. Let $e_i = \{c_{i1}, c_{i2}, ..., c_{im}\}$ and $e_j = \{c_{j1}, c_{j2}, ..., c_{jn}\}$ (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 is $e_i \cup e_j$. Else, we define $e'_i = \{c_k | c_k \in e_i \; and \; \{c_k\} \cup e_j \; is \; consistent\}$ and the resulting cumulative effect to be $e'_i \cup e_j$. 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 following, we shall use $acc(e_1, e_2)$ to denote the result of pair-wise effect accumulation of two contiguous tasks t_1 and t_2 with effects e_1 and e_2.

In addition to the effect annotation of each task, we annotate each task t with a cumulative effect E_t. E_t is defined as a set $\{es_1, es_2, ..., es_p\}$ of alternative effect scenarios. Alternative effect scenarios are introduced by OR-joins or XOR-joins, as we shall see below. Cumulative effect annotation involves a left-to-right pass through a sequence of tasks. Tasks which are not connected to any preceding task via a control flow link are annotated with the cumulative effect $\{e\}$ where e is the immediate effect of the task in question. We accumulate effects through a left-to-right pass of a sequence, applying the pair-wise effect accumulation procedure on contiguous pairs of tasks. The process continues without modification over splits. Joins require special consideration. In the following, we describe the procedure to be followed in the case of 2-way joins only, for brevity. The procedure generalizes in a straightforward manner for n-way joins.

AND-joins: Let t_1 and t_2 be the two tasks immediately preceding an AND-join. Let their cumulative effect annotations be $E1 = \{ec_{11}, ec_{12}, ..., ec1m\}$ and $E2 = \{ec_{21}, ec_{22}, ..., ec_{2n}\}$ respectively (where ec_{sc} denotes an effect clause within an effect scenario). Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the AND-join. We define $E = \{acc(ec_{1i}, e) \cup acc(ec_{2j}, e) | ec_{1i} \in E_1 and ec_{2j} \in E2\}$. Note that we do not consider the possibility of a pair of effect scenarios ec_{1i} and ec_{2j} being inconsistent, since this would only happen in the case of intrinsically and obviously erroneously constructed process models. The result of effect accumulation in the setting described here is denoted by $ANDacc(E_1, E_2, e)$.

XOR-joins: Let t_1 and t_2 be the two tasks immediately preceding an XOR-join. Let their cumulative effect annotations be $E1 = \{ec_{11}, ec_{12}, ..., ec_{1m}\}$ and $E2 = \{ec_{21}, ec_{22}, ..., ec_{2n}\}$ respectively. Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the XOR-join. We define $E = \{acc(ec_i, e) | ec_i \in E1 or eci \in E2\}$. The result of effect accumulation in the setting described here is denoted by $XORacc(E_1, E_2, e)$.

OR-joins: Let t_1 and t_2 be the two tasks immediately preceding an OR-join. Let their cumulative effect annotations be $E1 = \{ec_{11}, ec_{12}, ..., ec_{1m}\}$ and $E2 = \{ec_{21}, ec_{22}, ..., ec_{2n}\}$ respectively. Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the OR-join. The result of effect accumulation in the setting described here is denoted by $ORacc(E_1, E_2, e) = ANDacc(E_1, E_2, e) \wedge XORacc(E_1, E_2, e)$. Henceforth in our paper, for simplicity, we consider that OR-joins can be represented via XOR-joins themselves.

Pair-wise effect accumulation as described above will form the basis of our business process derivation algorithm, and will enable our algorithm to verify whether the derived business process design does meet the stated goals.

4 Goal Refinement and Constraint Specification

4.1 Goal Refinement

We define the *goals* of a business process as a combination $G_1 \wedge G_2 \wedge \ldots \wedge G_n$ of boolean conditions in CNF, all of which need to be satisfied at the end of the process execution. For example, the goal 'Link to Existing problem' of our running example in Fig. 1 can be represented as follows: *Goal: Achieve[LinkIncidentToProblemTicket] ($\forall$ i: incident, pt: problem ticket, p: problem, it: incident ticket) IsCausedBy(p,i) $\Rightarrow$ link(it, pt).*

Each boolean condition G_i can itself be broken down into a (conjunctive as well as disjunctive) combination of clauses, each of which is a sub-goal of G_i. The case of conjunction is best illustrated by the goal 'Incident and Problem Management' as it is a combination of sub goals 'Fix Problem', 'Diagnose Problem' and 'Verify Problem' and we expect all these sub goals to be realized in conjunction. Similarly the disjunctive case is illustrated again by the goal 'Diagnose Problem' as realizing this goal can satisfy one of the goals 'Link to existing problem' or 'Enrich Problem'.

Our goal refinement procedure is based on the KAOS methodology [9]. For a goal G_i, let it be expressed as $G_{i1} \wedge G_{i2} \ldots \wedge G_{im}$, where each sub-goal G_{ij} is of the form $G_{ij1} \vee G_{ij2} \ldots \vee G_{ijl}$. That is, each clause G_{ij} is a purely disjunctive clause. In accordance with [9], we say that the sub-goals *refine* G_i if the following hold:

1. $G_{i1} \wedge G_{i2} \ldots \wedge G_{im} \vdash G_i$ (entailment)
2. $\forall i : \wedge_{j \neq i} G_{ij} \nvdash G_i$ (minimality)
3. $G_{i1} \wedge G_{i2} \ldots \wedge G_{im} \nvdash false$ (consistency)

That is, the set of sub-goals for a goal will achieve the goal (entailment); it will be the smallest set of sub-goals to achieve the goal (minimality); and it will never be incorrect (consistency).

In its turn, each disjunctive clause can itself be refined into a collection of one or more conjunctive clauses, each of which could themselves possess a collection of two or more disjunctive clauses, and so on. Indeed, the presence of a disjunctive clause signifies a set of mutually exclusive options by which the particular sub-goal is to be satisfied. Later in Section 5, we will show how these clauses can be used to design XOR-splits and joins.

Our goal refinement procedure, therefore, refines the overall goals of a business process alternatively using conjunctive and disjunctive clauses, until all sub-goals have been completely specified to the user's satisfaction. The goal model that we presented in Fig. 1, is the outcome of such an exercise.

We define a *singleton* clause in a (refined) goal specification as a clause that is a single literal. In contrast, a non-singleton clause is a disjunctive combination $L_1 \vee L_2 \ldots \vee L_n$ of literals. The relevance of this distinction will become clear in Section 5.1, when we present our business process derivation algorithm.

4.2 Domain Constraint Specification

As we have seen, a goal is merely a collection of boolean conditions, without any specific ordering on how they have to be fulfilled in the business process. In case the analyst desires to impose an ordering, he/she can specify them in the form of what we call *domain constraints*, which are restrictions on the way in which the goal conditions are to be achieved. Business compliance constraints [1–3] can also be specified using our domain constraint formalism.

Formally, we define a domain constraint as a tuple $< \mathcal{C}_i, \mathcal{C}_j, rel >$; where $\mathcal{C}_i$ and $\mathcal{C}_j$ are boolean conditions; and rel is one of the following relations - IMM standing for *immediately*, EVE standing for *eventually*. This is to be interpreted as: the condition $\mathcal{C}_i$ has to be realized in the business process before the condition $\mathcal{C}_j$ can be realized. The operator rel qualifies this constraint by specifying how soon $\mathcal{C}_j$ should be realized after $\mathcal{C}_i$. Please note that our domain constraints are at a higher level of abstraction than task precedence constraints expressible in languages such as concurrent transaction logic (see [7] and the citations contained therein); indeed, later in Section 5.1 we will show how these constraints are used to create precedence constraints among the derived business process steps.

From the above formulation of domain constraints, the following proposition can be stated.

Proposition 1. *Any precondition can be represented via a domain constraint.*

Proof: If a task T_i has no predecessors, then it is the starting task of a business process, and its precondition can be represented via the domain constraint $< precondition(T_i), effect(T_i), IMM >$. If T_i has at least one predecessor, then its precondition can be represented as a boolean condition $C_1 \wedge C_2 \wedge \ldots \wedge C_n$, where each C_k is an effect of a predecessor. If the effect of T_i is $effect(T_i)$, then the precondition of T_i can be represented via the set of domain constraints $\{< C_k, effect(T_i), IMM > \}, 0 \leq k \leq n$. **QED**

In our running example, considering the goal 'Try Potential Fixes', we can construct the domain constraint as: $< IsProblemIsolated(it) \wedge AreKnownFixesAvaliable(it)$, $TryPotentialFixes(it)$, $EVE >$, where *it* denotes the incident ticket, $IsProblemIsolated(it)$ and $AreKnownFixesAvailable(it)$ as boolean conditions share the relation EVE with $TryPotentialFixes(it)$, again another boolean condition. Similarly the condition 'CanCreateNewProblem' has to be realized after the condition 'CannotEscalate' in realizing the goal 'CreateNewProblem' and they share the relation IMM.

5 Process Derivation from Goals

For deriving a business process design from goals and constraints, we assume the following inputs: a set of effect-annotated tasks in a capability library; a set of goals and sub-goals, refined until the level of an ontological match with the effects of the tasks in the capability library; and a set of domain constraints.

5.1 Process Derivation Algorithm

Our process derivation algorithm takes as input the refined (i.e., ontologically matching with effects) goals G & domain constraints DC, and effect-annotated tasks in the capability library, and produces a set of effect-annotated process steps PS and a set of *precedence constraints* PREC among the process steps. A precedence constraint among two process tasks $T_i \overset{rel}{\rightarrow} T_j$ specifies the order in which each task should execute vis-a-vis the other, and where $rel \in \{IMM, EVE\}$, with IMM standing for $immediately$ and EVE standing for $eventually$. The former type of precedence specifies that T_j must execute immediately after T_i has executed, whereas the latter specifies that T_j can executed any time after T_i has executed.

Our algorithm consists of the following steps. First, it distinguishes between singleton and non-singleton clauses; for each singleton clause, it identifies the appropriate task in the capability library whose effect entails the clause, and adds that task to the set of process steps PS. For each non-singleton clause, however, the algorithm determines a collection of tasks whose collective effects entail the clause. It then adds the tasks to PS, along with an appropriate XOR gateway. Second, the algorithm generates precedence constraints from the domain constraints. Third, the algorithm evaluates the generated precedence constraints for inconsistencies and alerts the user in case it discovers any, so that the user can resolve the inconsistencies. Finally, the algorithm generates a business process design from the (user-resolved) precedence constraints.

Precedence Constraint Generation: For now, we consider the sub-case when both $\mathcal{C}_i$ and $\mathcal{C}_j$ can be fulfilled by single tasks; the sub-case when either $\mathcal{C}_i$ or $\mathcal{C}_j$ is fulfilled by a disjunctive combination of tasks, is dealt with under XOR gateways.

Hence our algorithm for generating precedence constraints from the domain constraint $< \mathcal{C}_i, \mathcal{C}_j, rel >$, with each condition represented by a single task, works as follows. First, each condition $\mathcal{C}_i$ and $\mathcal{C}_j$ is analyzed, and the appropriate process tasks that fulfil the condition, are identified. Second, for each pair of tasks T_i, T_j, with T_i (resp. T_j) pertaining to $\mathcal{C}_i$ (resp. $\mathcal{C}_j$), the precedence constraint $T_i \overset{rel}{\rightarrow} T_j$ is generated, where rel is represented by EVE or IMM.

XOR Gateway Generation: We represent a disjunctive clause via an XOR gateway. In addition, we also need to accommodate domain constraints of the form $< \mathcal{C}_i, \mathcal{C}_j, rel >$, where either $\mathcal{C}_i$ or $\mathcal{C}_j$ is a disjunctive clause, and where one of either $\mathcal{C}_i$ or $\mathcal{C}_j$ is a non-singleton clause. This is needed in order to generate the appropriate precedence constraints from these domain constraints. Hence if such a domain constraint exists, we have three sub-cases:

1. Only $\mathcal{C}_i$ is a disjunctive clause: $\mathcal{C}_i$ would be represented via an XOR gateway $T_{i1} \vee T_{i2} \ldots T_{im}$, by tasks $T_{i1} \ldots T_{im}$ that collectively fulfill condition $\mathcal{C}_i$; and $\mathcal{C}_j$ by the single task T_j that fulfills condition $\mathcal{C}_j$. For this sub-case, we create a "dummy" XOR-join node $T_{i,m+1}$, whose effect is $\mathcal{C}_i$; and we create the following precedence constraints: $T_{ik} \overset{IMM}{\rightarrow} T_{i,m+1}, k = 1, \ldots, m$, and $T_{i,m+1} \overset{rel}{\rightarrow} T_j$.
2. Only $\mathcal{C}_j$ is a disjunctive clause: this is the reverse of the above sub-case; if $\mathcal{C}_j$ is represented via the XOR gateway $T_{j1} \vee T_{j2} \ldots T_{jm}$, and $\mathcal{C}_i$ by the task T_i, then the precedence constraints, $T_i \overset{rel}{\rightarrow} T_{jk}, k = 1, \ldots, m$, are generated.
3. Both $\mathcal{C}_i$ and $\mathcal{C}_j$ are disjunctive clauses: Let $\mathcal{C}_i$ be represented by an XOR gateway with p_i paths, and $\mathcal{C}_j$ be represented by an XOR gateway with p_j paths. Then, as in the first sub-case above, a "dummy" XOR-join node $T_{i,m+1}$, is first generated, whose effect is $\mathcal{C}_i$. Next, for each node $T_{ik}, 0 \leq k \leq j$ on the XOR gateway whose effect is $\mathcal{C}_j$, the following precedence constraint is generated: $T_{i,m+1} \overset{rel}{\rightarrow} T_{ik}$, $0 \leq k \leq j$.

For instance, our running example shows different types of problems based on their escalation support; escalation/Non-escalation cases would therefore differ. Hence there are 4 possible branches in the above scenario just based on the support for escalation at a given level. The user can later tweak the XOR gateway by pruning the variables, and thereby, the number of branches. However, that is beyond the scope of this algorithm.

Inconsistency Resolution & Business Process Design Generation: Once the precedence constraints are generated, inconsistencies could arise. For any pair of tasks T_i and T_j, we define an *inconsistency* as the existence of two precedence constraints that are mutually conflicting. That is, if there are two precedence constraints $T_i \overset{rel}{\rightarrow} T_j$ and $T_j \overset{rel}{\rightarrow} T_i$, where $rel \in \{IMM, EVE\}$, then this is an inconsistency. For each precedence constraint $T_i \overset{rel}{\rightarrow} T_j$, our inconsistency detection procedure checks whether there exists a (direct or transitively obtained) constraint $T_j \overset{rel}{\rightarrow} T_i$. Inconsistencies are flagged to the user, who will then need to resolve them manually. (We will be investigating automated inconsistency resolution for future work.)

The actual generation of the business process design, assumes that all inconsistencies have been resolved by the user. It basically consists of adding edges between tasks T_i and T_j based on the derived precedence constraints, whether t_i is supposed to *immediately* or *eventually* precede T_j. For the former case, the edge between two tasks is added right away. For the latter, on the other hand, we first check whether a chain of *immediately*-type constraints already exists on a path between the tasks. If so, then the last task on this chain is made the predecessor of T_j. If not, then T_i itself is made T_j's predecessor. While doing so, the algorithm also uses the effect accumulation procedure described in Section 3 to verify the compatibility of the business process under generation with the refined goals.

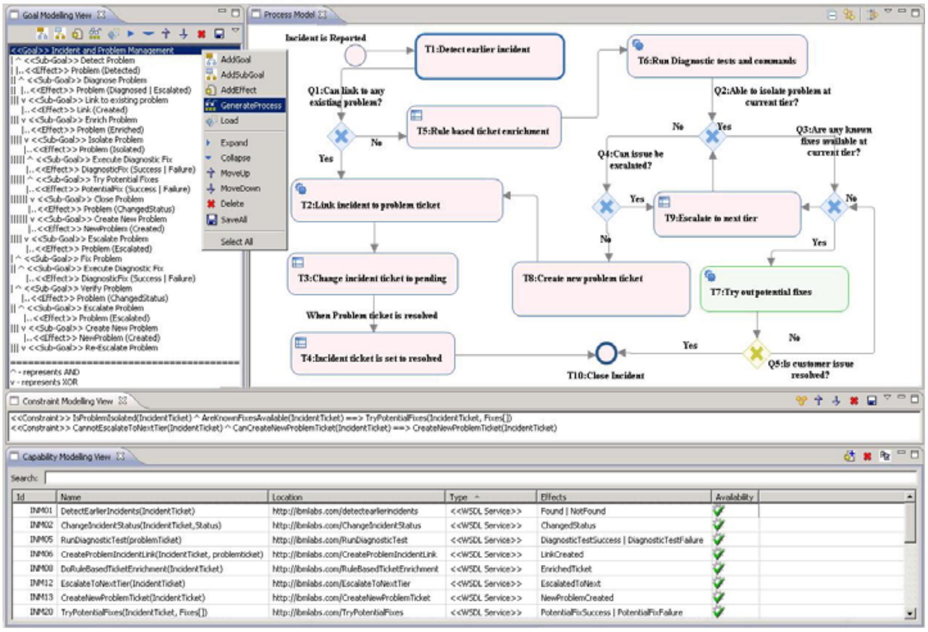
6 Prototype Implementation

Our prototype implementation is built as a plugin on IBM's Rational Software Architect (RSA) tool, and is depicted in Fig. 2. It was tested on a PC with 3.2 GHz processor speed and 3 GB of RAM. For our running example, once the user recorded the goals and domain constraints, the business process was generated within one minute.

The plugin provides the business analyst with various views that help define the inputs to business process derivation. The Goal Modeling view provides options to define the goals and sub-goals. Expected effect outcomes can be added as annotations for the respective goals and sub-goals that are represented in the AND/XOR logic. The domain constraints are defined in the Constraint Modeling view, which also helps generate the precedence constraints. The Capability Modeling view helps add various capabilities that can be used in order to fulfill the goals. Based on the capability availability and matching of capability effects with that of the specific goal/sub-goal in question, the goal-capability matching is arrived at. This helps in derivation of the incident management business process in BPMN format, which is also depicted in Fig. 2.

7 Related Work

Planning-based Business Process Derivation: The work reported in this paper is inspired in part by planning techniques from AI [4], in particular, partial-order planning. However, as we have already shown, such techniques require non-trivial adjustments in order to make them usable for business process derivation. Our business process derivation technique is also inspired in part by our earlier work [13], where we proposed a technique to map user's process goals into scenario descriptions described in the form of sequence diagrams. Appropriate composition of the sequence diagrams yields the final business process design.


Goal Modelling View
AddGoal
AddSubGoal
AddEffect
GenerateProcess
Load
Expand
Collapse
MoveUp
MoveDown
Delete
SaveAll
Select All
Process Model
Incident is Reported
T1:Detect earlier incident
Q1:Can link to any existing problem?
No
Yes
T5:Rule based ticket enrichment
T2:Link incident to problem ticket
T3:Change incident ticket to pending
When Problem ticket is resolved
T4:Incident ticket is set to resolved
T6:Run Diagnostic tests and commands
Q2:Able to isolate problem at current tier?
No
Yes
Q3:Are any known fixes available at current tier?
Q4:Can issue be escalated?
Yes
T9:Escalate to next tier
No
No
Yes
T8:Create new problem ticket
T7:Try out potential fixes
Yes
No
T10:Close Incident
Q5:Is customer issue resolved?
Constraint Modelling View
Capability Modelling View
Search:

```
<<Goal>> Incident and Problem Management
| ^ <<Sub-Goal>> Detect Problem
| |..<<Effect>> Problem (Detected)
|| ^ <<Sub-Goal>> Diagnose Problem
||  |..<<Effect>> Problem (Diagnosed | Escalated)
||| v <<Sub-Goal>> Link to existing problem
   |..<<Effect>> Link (Created)
||| v <<Sub-Goal>> Enrich Problem
   |..<<Effect>> Problem (Enriched)
|||| v <<Sub-Goal>> Isolate Problem
    |..<<Effect>> Problem (Isolated)
||||| ^ <<Sub-Goal>> Execute Diagnostic Fix
     |..<<Effect>> DiagnosticFix (Success | Failure)
||||| ^ <<Sub-Goal>> Try Potential Fixes
     |..<<Effect>> PotentialFix (Success | Failure)
|||||| v <<Sub-Goal>> Close Problem
      |..<<Effect>> Problem (ChangedStatus)
|||||| v <<Sub-Goal>> Create New Problem
      |..<<Effect>> NewProblem (Created)
|||| v <<Sub-Goal>> Escalate Problem
     |..<<Effect>> Problem (Escalated)
| ^ <<Sub-Goal>> Fix Problem
|| ^ <<Sub-Goal>> Execute Diagnostic Fix
     |..<<Effect>> DiagnosticFix (Success | Failure)
| ^ <<Sub-Goal>> Verify Problem
   |..<<Effect>> Problem (ChangedStatus)
|| ^ <<Sub-Goal>> Escalate Problem
    |..<<Effect>> Problem (Escalated)
||| v <<Sub-Goal>> Create New Problem
    |..<<Effect>> NewProblem (Created)
||| v <<Sub-Goal>> Re-Escalate Problem
==========================================
^ - represents AND
v - represents XOR
```

```
<<Constraint>> IsProblemIsolated(IncidentTicket) ^ AreKnownFixesAvailable(IncidentTicket) ==> TryPotentialFixes(IncidentTicket, Fixes[])
<<Constraint>> CannotEscalateToNextTier(IncidentTicket) ^ CanCreateNewProblemTicket(IncidentTicket) ==> CreateNewProblemTicket(IncidentTicket)
```

Id	Name	Location	Type	Effects	Availability
INM01	DetectEarlierIncidents(IncidentTicket)	http://ibmlabs.com/detectearlierincidents	<<WSDL Service>>	Found | NotFound	✓
INM02	ChangeIncidentStatus(IncidentTicket,Status)	http://ibmlabs.com/ChangeIncidentStatus	<<WSDL Service>>	ChangedStatus	✓
INM05	RunDiagnosticTest(problemTicket)	http://ibmlabs.com/RunDiagnosticTest	<<WSDL Service>>	DiagnosticTestSuccess | DiagnosticTestFailure	✓
INM06	CreateProblemIncidentLink(IncidentTicket, problemticket)	http://ibmlabs.com/CreateProblemIncidentLink	<<WSDL Service>>	LinkCreated	✓
INM08	DoRuleBasedTicketEnrichment(IncidentTicket)	http://ibmlabs.com/RuleBasedTicketEnrichment	<<WSDL Service>>	EnrichedTicket	✓
INM12	EscalateToNextTier(IncidentTicket)	http://ibmlabs.com/EscalateToNextTier	<<WSDL Service>>	EscalatedToNext	✓
INM13	CreateNewProblemTicket(IncidentTicket)	http://ibmlabs.com/CreateNewProblemTicket	<<WSDL Service>>	NewProblemCreated	✓
INM20	TryPotentialFixes(IncidentTicket, Fixes[])	http://ibmlabs.com/TryPotentialFixes	<<WSDL Service>>	PotentialFixSuccess | PotentialFixFailure	✓

Fig. 2. Prototype Implementation

The citations [14–16] describe techniques for semantic annoations of business processes using mining techniques, with applications such as adaptation and token analysis to identify components in business processes. While undoubtedly powerful, these techniques lack the simplicity of our semantic annotation approach. However, we will be investigating the adaptation application from [15] for future work.

Goal Modeling and Decomposition: The primary goal decomposition methodology that we have leveraged in this paper is KAOS [9], which provides a language and method for goal-driven requirements elaboration.

Business Process Compliance: Business process compliance management [2, 3] involves several aspects, of which the following are relevant for this paper, viz., verifying process compliance against compliance requirements, and business process derivation to ensure adherence to compliance requirements. For the former aspect, various frameworks [2, 11] have been developed to manage and check the violation of compliance policies by a given business process at design time, in order to minimize the cost of non-compliance. The citation [3] presents a semi-automated approach to synthesize business process templates out of compliance requirements expressed in linear temporal logic; however, that paper only focuses on compliance requirements, whereas our approach also considers functional requirements expressed as goals.

8 Future Work

Future work will involve testing our approach on larger case studies, and incorporating automated inconsistency resolution, process adaptation and incremental redesign.

References

1. Governatori, G., Rotolo, A.: A conceptually rich model of business process compliance. In: APCCM, pp. 3–12 (2010)
2. Governatori, G., Milosevic, Z., Sadiq, S.W.: Compliance checking between business processes and business contracts. In: EDOC, pp. 221–232 (2006)
3. Awad, A., Goré, R., Thomson, J., Weidlich, M.: An Iterative Approach for Business Process Template Synthesis from Compliance Rules. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 406–421. Springer, Heidelberg (2011)
4. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall (2009)
5. Henneberger, M., Heinrich, B., Lautenbacher, F., Bauer, B.: Semantic-based planning of process models. In: Multikonferenz Wirtschaftsinformatik (2008)
6. Heinrich, B., Bolsinger, M., Bewernik, M.: Automated planning of process models: The construction of exclusive choices. In: ICIS, paper 184 (2009)
7. Mukherjee, S., Davulcu, H., Kifer, M., Senkul, P., Yang, G.: Logic based approaches to workflow modeling and verification (2003)
8. Hinge, K., Ghose, A.K., Koliadis, G.: Process seer: A tool for semantic effect annotation of business process models. In: EDOC, pp. 54–63 (2009)
9. Darimont, R., van Lamsweerde, A.: Formal refinement patterns for goal-driven requirements elaboration. SIGSOFT Software Engineering Notes 21, 179–190 (1996)
10. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. 20(1-2), 3–50 (1993)
11. Ghose, A., Koliadis, G.: Auditing Business Process Compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)
12. Carbonell, J., et al.: Context-based machine translation. In: Proceedings of the 7th Conference of the Association for Machine Translation in the Americas, pp. 19–28 (2006)
13. Narendra, N.: A goal-based and risk-based approach to creating adaptive workflow processes. In: AAAI Spring Symposium on Bringing Knowledge to Business Processes (2000)
14. Lautenbacher, F., Bauer, B., Forg, S.: Process mining for semantic business process modeling. In: Enterprise Distributed Object Computing Conference Workshops, EDOCW 2009, September 13, pp. 45–53 (2009)
15. Lautenbacher, F., Eisenbarth, T., Bauer, B.: Process model adaptation using semantic technologies. In: Enterprise Distributed Object Computing Conference Workshops, EDOCW 2009, September 13, pp. 301–309 (2009)
16. Gotz, M., Roser, S., Lautenbacher, F., Bauer, B.: Token analysis of graph-oriented process models. In: Enterprise Distributed Object Computing Conference Workshops, EDOCW 2009, September 13, pp. 15 –24 (2009)