Semantic Monitoring and Compensation in Socio-technical Processes

Yingzhi Gou[1], Aditya Ghose[1,2], Chee-Fon Chang[1], Hoa Khanh Dam[2], and Andrew Miller[1]

[1] Centre for Oncology Informatics
Illawarra Health & Medical Research Institute
University of Wollongong, Australia
{yg452,c03}@uowmail.edu.au
[2] Decision Systems Lab.
School of Computer Science and Software Engineering
University of Wollongong, Australia
{aditya,hoa}@uow.edu.au

Abstract. Socio-technical processes are becoming increasingly important, with the growing recognition of the computational limits of full automation, the growth in popularity of crowd sourcing, the complexity and openness of modern organizations etc. A key challenge in managing socio-technical processes is dealing with the flexible, and sometimes dynamic, nature of the execution of human-mediated tasks. It is well-recognized that human execution does not always conform to predetermined coordination models, and is often error-prone. This paper addresses the problem of semantically monitoring the execution of socio-technical processes to check for non-conformance, and the problem of recovering from (or compensating for) non-conformance. This paper proposes a semantic solution to the problem, by leveraging semantically annotated process models to detect non-conformance, and using the same semantic annotations to identify compensatory human-mediated tasks.

1 Introduction

Socio-technical processes, which are executed by synergistic combinations of humans and technological components, have a long history, but have assumed new significance with the current interest in issues such as crowd-sourcing, human computation and gamification. They have also become important as a consequence of the introduction of process automation in settings where human-mediated functionality is critical and indispensable (such as clinical process management, military command and control, or air traffic management). An important aspect of socio-technical processes is that the human-mediated components are fallible, while the machine-mediated components are generally not (although there are critical exceptions). One way in which such fallibility might be manifested is via *structural non-conformance*, where activities are overlooked or executed in the wrong order, or where the wrong activities are executed.

M. Indulska and S. Purao (Eds.): ER Workshops 2014, LNCS 8823, pp. 117–126, 2014.

There is a mature body of work focusing on structural non-conformance (see [1] for a representative reference). Our focus is on the harder problem of *semantic non-conformance*, where we are interested in managing situations where the execution of a process might be structurally correct (the right activities are executed in the right order), but the effects achieved do not conform to what is required by design, potentially due to human errors. For instance, a clinical process might require the administration of an anti-hypertensive medication. The correct execution of this task would require that a nurse should deliver the medication to the patient in question and depart only when the patient has ingested the medication. A semantically non-conformant execution might occur if the nurse delivers the medication to the patient, but does not stay around to confirm that the patient has actually taken it (and the patient happens to not take the medication). In a hospital with a process-aware information system, the nurse might then confirm to the process engine that this task has been completed, leading to a situation where no structural non-conformance would be detected. The fact that this process instance is semantically non-conformant can only be determined by checking the *effects* of the process to ensure that what is expected is actually obtained. Thus, in our example, a blood pressure check later in the day might reveal elevated readings, when the expected readings are lower. This paper addresses the problem of *semantic monitoring* of socio-technical processes, by leveraging process designs that have been annotated with the expected effects at each point. Semantic non-conformance is flagged in settings where the observed effects deviate from the expected ones.

The human-mediated components of socio-technical processes also offer greater flexibility in "fixing" semantic non-conformance via the introduction of human-mediated activities constructed on the fly (generating new machine-mediated functionality, such as a new web service, can often take too long to be able to correct errors in an executing process instance). Thus, in our example, the semantic non-conformance detected via the blood pressure check can be fixed by having the nurse correctly administer the medication as soon as possible. Once this is done, the clinical process instance involving this patient would be restored to a semantically conformant state. This paper also addresses the problem of computing the best "fixes" of this kind, which we shall refer to as *compensations*. The problem is non-trivial. While the common-sense compensation in our running example might be to administer the anti-hypertensive medication as soon as the elevated blood pressure is detected, this might not be possible because of potential interactions between the anti-bypertensive medication and a more recently administered drug. We might thus need to search through the space of possible process re-designs to identify one where the earliest compensation is possible.

In the rest of this paper, we show how semantic annotation of process designs can be leveraged for a machinery for monitoring process execution based on effects (Section 2). We then formalize the notion of compensation and discuss a class of techniques that can be used to compute "optimal" compensations to deal with semantic non-conformance (Section 3). We describe the implementation and empirical evaluation of one of these techniques, with promising results (Section 4).

2 Semantic Process Monitoring

There is a large body of work that explores the use of semantic annotation of business process designs [2,3,4,5,6,7,8,9,10]. A large body of work also addresses the problem of semantic annotation of web services in a similar fashion [11,12,13,14]. Common to all of these approaches is the specification of *post-conditions*, which is what we primarily leverage in defining inter-process relationships. For our purposes, two aspects of the post-conditions (or effects) are important. First, post-conditions should be sensitive to process context, i.e., the post-conditions of a task at a certain point in a process design should reflect not just the effects achieved by executing that task but also the accumulated effects of the prior tasks in the process design that have been executed. Second, non-determinism must be accommodated in relation to post-conditions.

A number of the process annotation approaches referred to above achieve contextualization of post-conditions by using a device originally used in AI planning - add-lists and delete-lists of effects. Others, such as [5] and [8], use a state update operator derived from the literature on reasoning about action. We adopt this approach. The need for permitting non-determinism in effects stems from two observations. First, in any process with XOR-branching, one might arrive at a given task via multiple paths, and the contextualized post-conditions achieved must be contingent on the path taken. Since this analysis is done at design time, we need to admit non-deterministic effects since the specific path taken can only be determined at run-time. Second, many state update operators generate non-deterministic outcomes, since inconsistencies (that commonly appear in state update) can be resolved in multiple different ways. Our approach assumes that each task/activity is annotated with post-conditions (in the implementation presented later, we shall assume them to be unique, as much of the literature does, but this can be easily generalized to admit non-deterministic post-conditions), which are contextualized via a process of *effect accumulation*. We shall assume that all tasks (and their post-conditions) are drawn from an *enterprise capability library*. In this approach, we are able to answer, for any point in a process design, the following question: what will have happened if the process executes up to this point? The answer is a mutually exclusive set of *effect scenarios*, any one of which might describe the actual state of affairs at that point in the execution of the process design. Additional detail on the specific effect annotation and accumulation machinery used in the implementation can be found in Section 4.

We note that when a process is in a state that is (partially) characterized by an effect scenario, the execution of the next task in the model, or the occurrence of the next event, can lead to a very specific set of effect scenarios, determined by the *state update operator* being used. In effect, the process model determines a transition system, which determines how the partial state description contained in an effect scenario evolves as a consequence of the execution/occurrence of the next task (event) specified in the model. We assign each effect scenario appearing in a semantically annotated process model a unique ID (thus if the same partial description applies to a process at different points in its design, it would be assigned a distinct ID at each distinct point). We can thus refer

to the *predecessors* (the effect scenarios that can lead to the current scenario via a single state update determined by the next task/event) and *successors* (the scenarios that can be obtained from the current scenario via a single state update determined by the next task/event) of each effect scenario with respect to the transition system implicitly defined by the process design. There are works that have been done on obtaining such effect scenario, such as in [5] and [10], which also suggest that due to different paths at gateways could be taken before a task in a process model, and/or other reasons, there could be multiple effect scenarios associated with the task.

Definition 1. *A* **semantically annotated process model** *P is a process model in which each activity or event is associated with a set of* effect scenarios*. Each effect scenario es is a 4-tuple $\langle ID, S, Pre, Succ\rangle$, where S is a set of sentences in the background language, ID is a unique ID for each effect scenario, Pre is a set of IDs of effect scenarios that can be valid* predecessors *in P of the current effect scenario, while Succ is a set of IDs of effect scenarios that can be valid successors in P of the current effect scenario.*

A semantically annotated process model is associated with a set of normative traces, each providing a semantic account of one possible way in which the process might be executed.

Definition 2. *A* **normative trace** *nt is a sequence $\langle \tau_1, es_1, \tau_2, \ldots es_{n-1}, \tau_n, es_n\rangle$, where*

- *$es_i \ldots, es_n$ are effect scenarios, and for each $es_i = \langle ID_i, S_i, Pre_i, Succ_i\rangle$, $i \in [2..n]$, it is always the case that $ID_{i-1} \in Pre_i$ and $ID_i \in Succ_{i-1}$;*
- *$es_n = \langle ID_n, S_n, Pre_n, \emptyset\rangle$ is the* final effect scenario*, normally associated with the end event of the process;*
- *$es_1 = \langle ID_1, S_1, \emptyset, Succ_1\rangle$ is the* initial effect scenario*, normally associated with the start event of the process;*
- *Each of $\tau_1, \ldots, \tau_n$ is either an event or an activity in the process.*

We shall refer to the sequence $\langle \tau_1, \tau_2, \ldots, \tau_n\rangle$ as the identity *of the trace nt.*

To simplify of the presentation later on, the *es* in the trace, from now, refers to S in the 4-tuple $\langle ID, S, Pre, Succ\rangle$ because ID, Pre, and $Succ$ are meta-information used only to construct normative traces.

Definition 3. *A* **semantic execution trace** *of a process P is a sequence $\langle \tau_1, o_1, \tau_2, o_2, \ldots, \tau_m, o_m\rangle$ where each τ_i is either a task or an event, and each o_i is a set of sentences in the background language that we shall refer to as an* observation *that describes (possibly incompletely) the state of the process context after each task or event. We shall refer to the sequence $\langle \tau_1, \tau_2, \ldots, \tau_m\rangle$ as the* identity *of the execution trace.*

Note that we do not require each τ_i to belong to the process design P to allow the possibility of actual executions being erroneous. We will, on occasion, refer to a semantic execution trace, simply as an execution trace.

Definition 4. *An execution trace* $et = \langle \tau_1, o_1, \ldots, \tau_m, o_m \rangle$ *is said to be* **non-conformant** *with respect to a semantically annotated process* P *if and only if any of the following hold: (1) There exists an* o_i *in* et *such that for all normative traces* $nt' = \langle \tau'_1, es_1, \ldots, \tau'_i, es_i, \ldots \rangle$ *for which the identity of* $\langle \tau_1, o_1, \ldots, \tau_i, o_i \rangle$ *is a prefix of its identity and* $o_j \models es_j$ *for each* $j = 1, \ldots, i-1$, $o_i \not\models es_i$ *(we shall refer to this as* weak semantic non-conformance*). (2) If we replace non-entailment with inconsistency in condition (1) above, i.e.,* $o_i \cup es_i \models \perp$, *we obtain* strong semantic non-conformance*. In each case, we shall refer to* τ_i *as the* violation point *in the process.*

We only deal with semantic non-conformance in structurally compliant process instances. In other words, we assume that the identity of every semantic execution trace of interest equals the identity of some normative trace of the process.

3 Semantic Compensation

In this section, we formalize the notion of compensation and outline some strategies for computing these. In the following, we will view process instances as semantic execution traces. We will assume that each process is associated with a *goal assertion* g.

Definition 5. *A process instance* $et = \langle \tau_1, o_1, \ldots, \tau_m, o_m \rangle$ *will be referred to as a* **semantically compensated instance** *of a (semantically annotated) process* P *if there exist* τ_i *and* τ_j *in* et*, with* $i < j$*, such that* τ_i *is a violation point, and there exists a normative trace* $nt = \langle \tau_1, es_1, \tau_2, \ldots es_{h-1}, \tau_h, es_h, \ldots, \tau_n, es_n \rangle$ *of* P *with an identity for which* $\langle \tau_1, \ldots, \tau_{j-1} \rangle$ *serves as a prefix, such that* $o_k \models es_l$ *for* $k = j, \ldots, m$ *and* $l = h, \ldots, n$*. As well, it must be the case that* $o_m \models g$*. We shall refer to* τ_j *as the* compensation point*. The compensation point must be a task and not an event.*

Definition 6. *Given a semantically compensated process instance* $et = \langle \tau_1, o_1, \ldots, \tau_m, o_m \rangle$ *of* P *with a compensation point* τ_j*, a* **compensation** *is a process design* P' *for which the completion of* τ_{j-1} *serves as the start event and* $\langle \tau_j, o_j, \ldots, \tau_m, o_m \rangle$ *is a valid normative trace. Every normative trace associated with* P' *must end in an effect scenario* es *such that* $es \models g$*, where* g *is the goal associated with the original process* P*.*

This definition of compensation is fairly general. More specifically, we are interested in *optimal compensations*, driven by the following intuitions. We prefer earlier compensations. In other words, we aim to ensure that as few system states as possible deviate for the normative process design (noting that a later compensation will necessarily mean that there would be more states between the violation point and the compensation point). We also prefer to minimize deviation of the overall semantically compensated process instance from the semantic "intent" of the original process design. These preferences can lead to competing pulls. We might in some situations be able to introduce an earlier compensation, but the compensation, while ensuring conformance from subsequent steps

(assuming no other steps deviate), might lead to greater changes in the system states than a potential later compensation.

Computing a compensation thus requires that we identify a process design which permits us to complete the currently executing process instance from the compensation point onwards in a manner that gives us a complete semantic execution trace that is as close as possible to the normative trace that would have been executed has there been no violation. The occurrence of a violation entails that we are only able to identify a prefix of this normative trace (the part that is actually executed prior to the violation). Given that multiple normative traces associated with the process design may share that prefix, we do not actually know which of these we would have actually executed had there been no violation. One way to compute the compensation is to identify that process design (or designs) which would minimize deviation from this set of normative traces (by picking one that minimizes the distance to the either the closest, or the farthest normative trace). This requires a distance measure to assess the distance between an execution trace and a normative trace. This distance measure must take into account both structural similarity (e.g., the number of activities in common between the two traces) and semantic similarity (e.g., the extent to which a set of observed assertions agree with an effect scenario). We describe an implementation with one such distance measure in the next section.

4 Implementation and Evaluation

In this section, we outline one specific implementation of the general framework for semantic process monitoring and compensation described above and present some preliminary empirical results. We note that the general framework could be instantiated in multiple ways (indeed the space of alternative design decisions is very large) and we do not suggest that this particular implementation is to be preferred to other possible ones (such claims can only be made after a series of substantive comparative studies). However, this particular implementation provides an adequate basis for making a preliminary determination of whether this approach is practical.

We use a machinery for semantic annotation of business process designs represented in BPMN. We omit details here for brevity but these can be found in [5]. It uses a syntactic state update operator based on the Possible Worlds Approach (PWA) [15]. The choice of this particular operator is mainly a matter of convenience (and adequate for assessing feasibility), while other operators, such as one based on the Possible Models Approach (leveraged by [8]) could also be used. We assume that a process model, semantically annotated using this machinery, is provided as input.

To measure the structural distance between a pair of sequences of activities/events, we use the *Levenshtein Distance* $lev(a, b)$ where $a = \langle a_1, \ldots, a_n \rangle$ and $b = \langle b_1, \ldots, b_m \rangle$.

For semantic distances, we define a simple distance function $\phi(es, o)$ where es is an effect scenario and o is a set of observations. We note that many, potentially

more sophisticated schemes for measuring semantic distance exist, but this is adequate for preliminary analysis. In the following, V_{strong} computes the number of assertions in an effect scenario that contribute to strong semantic non-conformance (as in Definition 3), while V_{weak} computes the number of assertions that contribute to weak semantic non-conformance. We leverage a background knowledge base KB that contains, amongst others, domain and compliance constraints.

$$V_{strong} = \{e | e \in es, o \cup \mathcal{KB} \models \neg e\}$$
$$V_{weak} = \{e | e \in es, o \cup \mathcal{KB} \not\models e, e \notin V_{strong}\}$$
$$\phi(es, o) = w_{strong} \times |V_{strong}| + w_{weak} \times |V_{weak}|$$

where, w_{strong} and w_{weak} are weights. If all observations reveal complete state descriptions, then weak violations do not apply. We can focus attention solely on strong or weak violations by appropriately setting the corresponding weights.

We measure the distance between a normative trace $nt = \langle a_1, es_1, \ldots, a_n, es_n \rangle$ and a semantic execution trace $et = \langle b_1, o_1, \ldots, b_m, o_m \rangle$ using the following function:

$$J(nt, et) = \sum_{i=1\ldots n} \min_{j=1\ldots m} (w_1 \times \phi(es_i, o_j) + w_2 \times lev(\langle a_1, \ldots, a_i \rangle, \langle b_1, \ldots, b_j \rangle))$$

where w_1 and w_2 are the weights for each distance.

Our prototype takes a semantically annotated business process and a capability library as inputs, then generates a set of all normative traces. We simulate a normative execution trace and randomly insert a violation in it. Once a violation is detected, the compensation computation machinery initiates a search for a sequence of activities from the capability library that can constitute a valid completion of the current partially complete process instance and that guarantees that it terminates in a goal-satisfying state. The prototype performs an exhaustive constructive search. Every candidate partial extension of the current process instance is evaluated for compliance with the KB. In the event of non-compliance, the search backtracks and evaluates an alternative extension. Our evaluation uses a propositional language for representing effects and the KB. Effect accumulation, goal satisfaction and compliance checking require the use of a theorem prover - in our prototype, the SAT4J SAT solver (modified to generate all maximal consistent subsets) is used for this purpose. We apply the effect accumulation machinery to generate a semantic trace from each of the valid task sequences identified by the search procedure. This gives us a set of semantically compensated process instances which are then ranked according to the nearest distance to a valid normative trace (i.e., for each process instance, we compute the shortest distance to any valid normative trace, and the instance with shortest distance amongst all appears at the top of the ranking, and so on). We limit each task in the capability library to be used only once in a semantically compensated process instance.

In the evaluation, we manually design 5 distinct semantically annotated process models with variations in the number of activities, gateways (we only use

Table 1. Evaluated Process Models

Process Model ID	Complexity of Process			Complexity of Semantic Annotation			Complexity of Knowledge Base	
	Total Number of Activities and Events	Length of Paths in the Model (Min/Max)	Number of Gateways (split and merge)	Size of Propositional Vocabulary	Length of Task Post-conditions (Min/Max)	Length of Effect Scenarios (Min/Max)	Number of Clauses in the Knowledge Base	Number of Activities in Capability Library
1	6	6/6	0	3	1/2	1/3	3	4
2	12	12/12	0	13	1/7	7/13	3	10
3	9	6/7	2	13	2/7	7/13	9	7
4	9	7/7	6	5	1/3	1/5	1	17
5	9	7/7	6	13	1/7	7/10	9	10

Table 2. Best Evaluation Result

Process Model ID	Location of Violation in Process	Shortest distance between process instance and normative trace	Goal Compliance	Time to compute best compensation(mm:ss:SSS)
1	Beginning	10	NO	00:00:199
1	Middle	10	NO	00:00:038
1	End	3	YES	00:00:085
2	Beginning	114	NO	10:00:206
2	Middle	30	NO	00:00:019
2	End	1	YES	00:14:817
3	Beginning	30	NO	04:53:580
3	Middle	30	NO	00:00:009
3	End	2	NO	00:00:034
4	Beginning	9	YES	01:55:569
4	Middle	6	YES	00:16:785
4	End	7	YES	00:04:986
5	Beginning	38	NO	00:00:010
5	Middle	17	NO	00:00:018
5	End	2	NO	00:00:102

XOR gateways), complexity of the knowledge base and effect scenarios etc (note that these cannot be randomly generated). These dimensions of the 5 process model are summarized in Table 1. We then identify the quality of solutions generated within a 10 minute time bound and report these results in Table 2. The table only shows summaries of the best compensated process instances from multiple runs of evaluation (violations are randomly generated).

Analysis of the results: The results we obtain here are only modestly encouraging. We note that none of the minimum distances for the compensated process instances are 0, but this is not a negative (any violation will lead to a non-zero distance). The location the violation is clearly important. A violation at the beginning of a process presents a much larger search space than a violation later in the process. The more complex the semantic annotations are, the longer it takes to compute compensations (which is not surprising). Process models 4 and 5 are structurally identical, but 5 has semantic annotations that are significantly more complex than those of 4. As a result, we are able to compute a goal-satisfying compensation from process 4 within the time-bound, but not for process 5. In general, not all of the "closest" process instances are goal compliant. Many socio-technical processes of interest have durations far greater than 10 minutes, hence the fact that we are able to compute goal-satisfying compensations for many (if not all) of the processes is actually encouraging. This suggests that with a higher time-bound, we might find even better and more goal-satisfying compensations, while still being able to compensate quite early in these long-duration processes.

5 Related Work

Cook et al. [16] offer a process validation framework, which involves comparing the event stream from the process model against the event stream from the log using string distance metrics. Rozinat and van der Aalst [1] developed the Conformance Checker as part of the ProM framework which, given a process design and a collection of its event log from execution, determines whether the process execution behavior reflects the designed behavior. Different from [1] and [16], our semantic conformance checking assumes that the instance of executed process is structurally correct. A number of proposals for goal-oriented process management exist [17,18]. Klein and Dellarocas [19] present a knowledge-based approach to exception detection and handling in work-flow systems. They define an exception as "any deviation from an 'ideal' collaborative process that uses the available resources to achieve the task requirements in an optimal way" [19]. In their exception management approach, the participant of an enacted process will be notified when there is an exception with the exception types and associated exception handler processes proposed by the work-flow designer, so that the participants are able to modify the instance of the process to resolve the exception and allow the process to continue. Our approach does not require that exceptions handlers be written for every possible exception.

6 Conclusion

In this paper we present a novel framework for semantic monitoring and compensation of business processes, leveraging semantic annotations of process designs. We identify some abstract strategies for implementing such a framework, and then present a concrete implementation. The evaluation of the implementation suggests that there is modest room for optimism that such an approach would be viable in practice.

References

1. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)
2. Fensel, D., Facca, F., Simperl, E.: Web Service Modeling Ontology. In: Semantic Web Services, pp. 107–129. Springer (2011)
3. Fensel, D., Lausen, H., Polleres, A., Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer (2006)
4. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D.: Semantic business process management: A vision towards using semantic Web services for business process management. In: IEEE International Conference on e-Business Engineering, pp. 535–540. IEEE (2005)
5. Hinge, K., Ghose, A., Koliadis, G.: Process SEER: A tool for semantic effect annotation of business process models. In: Proceedings of the 13th IEEE International EDOC Conference. IEEE Computer Society Process (2009)

6. Di Pietro, I., Pagliarecci, F., Spalazzi, L.: Model checking semantically annotated services. IEEE Transactions on Software Engineering 38, 592–608 (2012)
7. Smith, F., Proietti, M.: Rule-Based Behavioral Reasoning on Semantic Business Processes. In: Proceedings of the 5th International Conference on Agents and Artificial Intelligence, pp. 130–143. SciTePress (2013)
8. Weber, I., Hoffmann, J., Mendling, J.: Beyond soundness: On the verification of semantic business process models. Distributed and Parallel Databases 27, 271–343 (2010)
9. Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., Tonella, P.: Semantcally-aided business process modeling. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 114–129. Springer, Heidelberg (2009)
10. 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)
11. Martin, D., et al.: Bringing semantics to web services: The OWL-S approach. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 26–42. Springer, Heidelberg (2005)
12. Meyer, H.: On the Semantics of Service Compositions. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524, pp. 31–42. Springer, Heidelberg (2007)
13. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographiess. ACM Transactions on the Web 4, 3:1–3:62 (2010)
14. Smith, F., Missikoff, M., Proietti, M.: Ontology-Based Querying of Composite Services. In: Ardagna, C.A., Damiani, E., Maciaszek, L.A., Missikoff, M., Parkin, M. (eds.) BSME 2010. LNCS, vol. 7350, pp. 159–180. Springer, Heidelberg (2012)
15. Ginsberg, M.L., Smith, D.E.: Reasoning about action I: A Possible World Approach. Artificial Intelligence 35(2), 165–195 (1988)
16. Cook, J.E., Wolf, A.L.: Software process validation: quantitatively measuring the correspondence of a process to a model. ACM Transactions on Software Engineering and Methodology 8(2), 147–176 (1999)
17. Ghose, A., Koliadis, G.: Actor eco-systems: From high-level agent models to executable processes via semantic annotations. In: Proceedings of the 31st Annual International Computer Software and Applications Conference, vol. 02, pp. 177–184. IEEE Computer Society, Washington, DC (2007)
18. Koliadis, G., Vranesevic, A., Bhuiyan, M., Krishna, A., Ghose, A.K.: A combined approach for supporting the business process model lifecycle. In: Proceedings of the, Pacific Asia Conference on Information Systems, pp. 1305–1319 (2006)
19. Klein, M., Dellarocas, C.: A knowledge-based approach to handling exceptions in workflow systems. Computer Supported Cooperative Work 9, 399–412 (2000)