

Verifying Semantic Business Process Models in Inter-operation

George Koliadis and Aditya Ghose
Decision Systems Laboratory
School of Computer Science and Software Engineering
University of Wollongong (UOW)
{gk56, aditya}@uow.edu.au

Abstract

Process inter-operation is characterized as cooperative interactions among loosely coupled autonomous constituents to adaptively fulfill system-wide purpose. Issues of inconsistency can be anticipated in inter-operating processes given their independent management and design. To reduce inconsistency (that may contribute to failures) effective methods for statically verifying behavioral interoperability are required. This paper contributes a method for practical, semantic verification of interoperating processes (as represented with BPMN models). We provide methods to evaluate consistency during process design where annotation of the immediate effect of tasks and sub-processes has been provided. Furthermore, some guidelines are defined against common models of inter-operation for scoping traceability to possible causes of inconsistency. This supports subsequent resolution efforts.

1. Introduction

The goal of system inter-operation is independent component evolution, while still maintaining each components ability to interact efficiently and conveniently [17]. Interoperation (between systems) is defined as “cooperative interactions among loosely coupled autonomous constituents to adaptively fulfill system-wide purpose” [3]. Additionally, [13] defines interoperability as the “effective capability of mutual communication of information, proposals and commitments, requests and results (including exceptions)”.

Successful inter-operation is characterized [3] as requiring: *common purpose; sufficient cooperation; minimal constrain; and, consistency of action* among constituents in order to achieve required product and service outcomes. These outcomes are enabled via “emergent effects [arising from cumulative action and interaction] that produce the de-

sired global properties in continuously changing situations” [3]. Changing situations may arise given: the autonomy of the inter-operating entities; the independent management and design of their activities; and, changing environmental influences (e.g. regulatory, cultural).

Current trends in business process inter-operation [13] are moving toward unified and federated approaches requiring rigorous support for: *preserving participant autonomy; failure tolerance / remediation; complex coordination between multiple existing* (or new partners); and *joint cross-organizational process management* at all architectural levels. Furthermore, increased support for life-cycle functions (i.e. operational, managerial and design) such as “facilities for statically verifying behavioral interoperability and preservation of information semantics in the collaboration” are in need [13].

In this work we focus on problems arising via inconsistent action. We propose a simple and practical approach for static design-time verification of business process model properties (represented in BPMN), annotated with additional semantic information in the form of *effects*. This allows us to reveal accidental (or deliberate) design / implementation errors that may adversely affect inter-operation. In most cases, these may not be identified due to the ‘high-level’ nature of most business process models, leading to implementation errors. In this approach, we cannot hope to anticipate all errors as we are dependent on the amount of information provided by analysts (as with other approaches). However, we believe many errors that may go unnoticed will be identified given the accumulation of information within models (discussed in Section 2).

1.1. Related Work

Existing approaches to model checking rely on detailed fine grained execution and interaction models [9] [10] [11] [6]. That is, when working with high-level models at an early phase of design (such as in the case of BPMN [26]),

adequate information with which to conduct effective model checking may not be available (i.e. designed primarily by business analysts in conversation with process participants). These models may not include a rich state description as modeled by Work-Flow Nets [22], and may include the integration of unbound (possibly nested) loops that may not guarantee the generation of finite state models. Furthermore, many techniques employing model checking have limited support for localizing errors and inconsistencies to specific (or range of) elements on process models.

In [23], the consistency of UML models are managed via an approach for [de]composing sound workflow nets (via synchronization and projection). These nets are used to model event transition in use cases and object lifecycles. Process verification / validation is applied to these integrated and detailed use case and object lifecycles.

In [25] describes an approach for checking semantic integrity constraints within the narrative structure of web documents. Description logic extensions to Computational Tree Logic (CTL) are provided for specifying a formal model of a documents conventions, criteria, structure and content. Analyst annotations are applied to describe the properties of specific document fragments with reference to some background knowledge. Thus, the approach provides the means to verify the semantic properties of a documents content along narrative paths rather than just transitions between pages.

In [14] open workflow nets are used to analyze interacting BPEL processes w.r.t. their controllability, and [27] propose a structure and approach to track processes across cross-organizational workflows. In addition, [15] propose a method for verifying semantic properties of a process w.r.t. execution traces once change operations have been applied, and [5] describe approaches to verify process integration operations. Finally, [12] describe an approach for developing UML sequence diagrams in a co-evolutionary manner with agent-oriented conceptual models by including semantic effect annotations.

Finally, Statecharts [7] provide a visual formalism for efficiently representing rich state-based system specifications. Statecharts however, do not easily lend themselves for direct use for verifying high-level processes, and as such we class our approach (in comparison) as a lightweight and approximate means for achieving this purpose.

Specifically, we aim to provide a means to verify inter-model properties when designed for inter-operation. As a result it may be used in a stand-alone or complementary manner with other techniques. Additionally, the information collected during annotation will be valuable in supporting anticipated implementation of system components that completely (or partially) automate process activities (e.g. the development of a data model). Furthermore, the generality of our approach, based on a standard set of modeling

elements (inc. basic control flow constructs), makes it applicable in a wide range of applications and notations (with close to minimal alteration). We describe our approach with a focus toward supporting interoperability considerations such as clarity of common purpose and managerial and design independence. We illustrate the approach using inter-operating business process models defined using the Business Process Modeling Notation (BPMN) [26]. Inconsistencies are identified between accumulated effects, annotated to state altering model elements.

Below, we provide a discussion of the business process modeling notation, and standard modes of inter-operation to be used. Section 2 then outlines our proposed model annotations and procedures. Section 3 provides illustration, prior to concluding in Section 4.

1.2. Business Process Modeling

The Business Process Modeling Notation (BPMN) [26] has received strong interest and support from tool vendors as an open standard for modeling business processes. BPMN is found to be of high maturity in representing the concepts required for modeling business process, apart from some limitations regarding the representation of process state and possible ambiguity of the swim-lane concept [1].

Processes are represented in BPMN using **flow objects**: *events*, *activities*, and *decisions*; **connecting objects**: *control flow links*, and *message flow links*; and **swim-lanes**: *pools*, and *lanes* within pools. BPMN allows three (3) degrees of inter-operation to be modeled. Firstly, *private* (i.e. or internal) business processes indicate no inter-operation with external entities and processes. In this case, single pool may be used to represent the organization or to delegate responsibility with swim-lanes. An *abstract* (i.e. or public) business process on the other hand may include multiple pools to support communication with abstract external entities (i.e. no explicit external processes are shown). In the third case, *collaboration* (i.e. or global) business processes (as in Figures 2, 3) explicitly define the internal behavior of all interoperating participants. Figure 3 illustrates a collaborative (inter-operational) Package Routing process.

1.3. Standard Modes of Interoperation

The Workflow Handbook [8] describes three main modes of interoperation that are applicable to our current context. Firstly, sequential (“chained” or “serial”) interoperation occurs over a single / synchronization point following a sequential and uni-directional flow of control and information between the two processes.

Next, nested inter-operation draws similarity with hierarchically structured sub-processes in that execution control is passed to a subsequent inter-operating process with block-

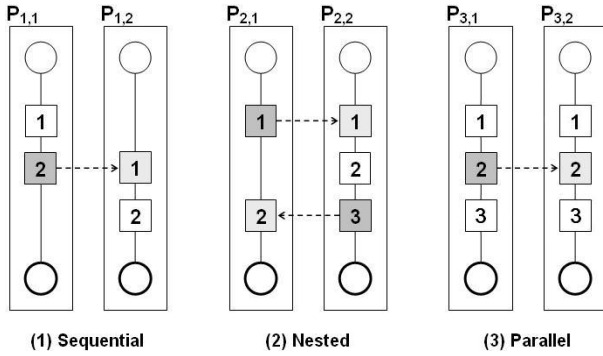


Figure 1. Models of Process Inter-operation.

ing on the execution of the first. Furthermore, “the hierarchic relationship may be continued across several levels, forming a set of nested sub-processes” [8].

In the parallel case, “two processes operate essentially independently. . . but requires that synchronization points exist between the two processes. . . once the processes each reach a predefined point” [8]. Therefore, corresponding activities are synchronized, based on control or information transfer requirements.

2. Using Model Annotations to Semantically Verify Models of Interoperating Processes

Activities and *Sub-Processes* (i.e. represented in BPMN as rounded boxes) signify a transition of state. Therefore, the labeling of an activity (e.g. ‘Register New Customer’) generalizes one or more normal/abnormal outcomes. In order to improve the clarity and descriptive capability of process models, we augment state altering nodes (i.e. atomic activities and sub-processes) with *effect annotations*. This parsimonious extension to the BPMN notation permits modelers to annotate activities in a process model with a richer specification of its immediate effects.

2.1. Effect Annotation

An *effect* is the result (i.e. product or outcome) of an activity being executed by some cause or agent. An *effect annotation* relates a specific result or outcome to an activity on a business process model. It explicitly states a result of the activity in its domain of execution. A causal relationship exists between a process activity and an effect. An activity can cause many effects, and an effect can be caused many activities. Effects can be viewed as both: *normative* - as they state required outcomes; and, *descriptive* in that they describe the normal, and predicted, subset of all possible outcomes.

When an analyst is annotating existing process models, the conditions labeling control-flows leaving a decision gateway may provide some understanding of the effect of a downstream activity. Effects may also refer to assumptions on how the immediate state of an observer (i.e. during inter-operation) may change as a result of some information / work item transfer. When implemented within a tool, effects may be viewed on a business process model graphically, or added to activity metadata.

We recommend that *informal annotations* of effect be applied as a first pass to ensure a rich expression of effects and for ease of communication. Effect annotations can be *formal* (for instance, in first order logic, possibly augmented with temporal operators), or *informal* (such as simple English). Many of the examples we use in this paper rely on formal effect annotations, but most of our observations hold even if these annotations were in natural language. For example, through the use of Controlled Natural Languages (CNL) with grammar and vocabulary restrictions such as in [16] [20] [4] [21]. Formal annotations (i.e. provided, or derived from CNL) permit us to use automated reasoners, while informal annotations oblige analysts to check for consistency between effects.

Effect annotations are formed in the indicative mood, or as fact (e.g. Provided(Courier, Contract, Details, Unsigned)). We have also employed some intuitive grammatical and vocabulary constraint to the example effects illustrated in Section 3.

- *Performs*(Agent, Action, Object) - This signifies an event that has occurred, which may be governed by constraints that are specified in a domain ontology.
- *Knows*(Agent, Object, Property, Value) - This predicate describes how a participants knowledge is updated via the enactment of an activity.

An *annotated BPMN model*, for the purposes of this paper, is one in which every task (atomic, loop, compensatory or multi-instance) and every sub-process has been annotated with descriptions of its immediate effects. We will now describe a procedure for accumulating these effect annotations to obtain a *cumulative effect annotation* for a complete process. We will assume that formal annotations are available in describing this procedure. 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 or CNF. Simple techniques exist for translating arbitrary sentences into the conjunctive normal form (e.g. [18]).

2.2. Effect Accumulation

We define a process for *pair-wise effect accumulation*, which, given an ordered pair of tasks with effect annota-

tions, determines the cumulative effect after both tasks have been executed in contiguous sequence.

Let $\langle t_i, t_j \rangle$ be the ordered pair of tasks, and let e_i and e_j be the corresponding pair of effect annotations. Let $e_i = \{c_{i1}, c_{i2}, \dots, c_{im}\}$ and $e_j = \{c_{j1}, c_{j2}, \dots, c_{jn}\}$ (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 $e'_i = \{c_k | c_k \in e_i \text{ and } \{c_k\} \cup e_j \text{ is consistent}\}$ 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 .

Effects are only accumulated within participant lanes. 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 $\{es_1, es_2, \dots, es_p\}$ 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 participant lane. 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 participant lane, applying the pair-wise effect accumulation procedure on contiguous pairs of tasks connected via control flow links. 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 $E_1 = \{ec_{11}, ec_{12}, \dots, ec_{1m}\}$ and $E_2 = \{ec_{21}, ec_{22}, \dots, ec_{2n}\}$ 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 $E = \{acc(ec_{1i}, e) \cup acc(ec_{2j}, e) | ec_{1i} \in E_1 \text{ and } ec_{2j} \in E_2\}$. 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 $E_1 = \{ec_{11}, ec_{12}, \dots, ec_{1m}\}$ and $E_2 = \{ec_{21}, ec_{22}, \dots, ec_{2n}\}$ 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 $E = \{acc(ec_i, e) | ec_i \in E_1 \text{ or } ec_i \in E_2\}$. 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 $E_1 = \{ec_{11}, ec_{12}, \dots, ec_{1m}\}$ and $E_2 = \{ec_{21}, ec_{22}, \dots, ec_{2n}\}$ 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 $ORacc(E_1, E_2, e) = ANDacc(E_1, E_2, e) \cup XORacc(E_1, E_2, e)$.

Take for example Figure 3, esp. the activities labeled ‘Identify Package’ (e_3), ‘Label Package’ (e_4), and ‘Update Status’ (e_5). The cumulative effect at ‘Label Package’ would be $E_4 = \{e_4 \cup e_n | e_n \subseteq e_3 \text{ and } e_4 \cup e_n \text{ is consistent}\}$. We would then determine the cumulative effect at e_5 to be either E_5 or E'_5 where $E_5 = \{e_5 \cup e_k | e_k \subseteq e_3 \text{ and } e_k \cup e_5 \text{ is consistent}\}$ and $E'_5 = \{e_5 \cup e_l | e_l \subseteq E_4 \text{ and } e_k \cup e_5 \text{ is consistent}\}$. That is, E_5 and E'_5 are possible results from two alternate paths that may be selected upon enactment. We will refer to the cumulative effect annotation of the final task in a process as the *cumulative effect of the process*.

2.3. Verifying the Consistency of Interoperating Business Processes Models

Inconsistencies in design artifacts exist when some domain / process specific rules required to hold between or within the artefact[s] are left unsatisfied at design-time. These rules may check for the *existence of* or *agreement between* information represented *within* or *among* design artefact[s] [2]. [24] state “a set of descriptions is inconsistent if there is no way to satisfy those descriptions all together”, and provide a broad inconsistency classification scheme.

Our focus in this paper is on semantic inconsistencies between inter-operating process specifications. We evaluate inconsistency between effects, during accumulation in a process, in order to ensure that the properties required of objects transferred between inter-operating processes are met. We define *inconsistency* in inter-operating business process models as a contradiction between the effect scenarios of two *synchronizing* tasks. The aforementioned accumulation procedure is effective in providing a local description of the

accumulated context within a process as well as the immediate effect of each activity. Therefore allowing us to conduct a pairwise analysis.

By using the aforementioned *Knows* predicate, we can include a *general rule* for identifying inconsistencies *during* synchronization. This rule states that agent (i.e. Participants) agreement (a_1, a_2) must exist between values (v_1, v_2) assigned to the properties of an object at the points of synchronization / information transfer in inter-operating processes. Formally,

- $\forall a_1, a_2 : Agent, o : Object, p : Property, v_1, v_2 : Value, Knows(a_1, o, p, v_1) \wedge \neg equal(v_1, v_2) \Rightarrow \neg Knows(a_2, o, p, v_2)$.

Let P_i and P_j be two inter-operating processes, and let $t_i \in P_i$ and $t_j \in P_j$ be two corresponding tasks (poss. in synchronization). Let their cumulative effects be $E_{t_i} = \{es_{i1}, \dots, es_{in}\}$ and $E_{t_j} = \{es_{j1}, \dots, es_{jm}\}$ respectively. Also let D represent some set of background knowledge that also includes the previous rules. An *inconsistency* exists between t_i and t_j , if for some effect scenarios $es_{in} \in E_{t_i}$ and $es_{jm} \in E_{t_j}$, $es_{in} \cup es_{jm} \cup D \vdash \perp$ holds. In other words es_{in} , es_{jm} and D *contradict* each other.

2.4. Identifying Possible Causes of Inconsistency in Inter-operating Processes

Once identified, inconsistencies can be localized between two tasks, participating in some object transfer during synchronization. That is, for some inconsistencies to exist there must be some disagreement between the *actual* and *expected* accumulated properties of two objects.

In order to resolve an inconsistency, analyst involvement will be required. Negotiation between the participants will need to occur, conceivably with reference to some agreed upon contract. In order to guide negotiations, an identification of the possible causes would be desirable. In Section 3 below we describe and illustrate (with examples) some rules that can be applied, once inconsistencies are found for identification of primary causes.

Specifically, inconsistencies between inter-operating processes have unique causes that can be identified via *regression analysis*. This involves backtracking through a process involved in an inconsistency to a point where the inconsistency does not hold. As a result, the preceding activity, where the inconsistency held is identified as responsible for introducing the inconsistency. In some cases, the cause may be traced across synchronization points to other inter-operating processes. The decision at this point as to the offending activity / process is also left to the analyst.

3. Detecting Inconsistencies in Interoperating Business Process Models

As will be discussed in the sections below, our approach permits static *design-time* verification in the sequential, nested and parallel cases. In *run-time* settings the capability to manage and schedule concurrent interleaving action (including roll-backs) would be required.

The following examples relate to the operations of a Transport Organization responsible for routing containers and packages between customers. Prior to verifying consistency, the effects of activities on the models being tested should be accumulated.

3.1. Sequential and Nested Inter-operation

During both sequential and nested process inter-operation, inconsistency will be simply identified across synchronization points. Intuitively, this means that some required property of a shared object (possibly being transferred between organizations) does not hold. Furthermore, the scope of sequential and nested inconsistencies can be traced to some activity prior to the activities that are synchronizing on either the source or target side of the synchronization.

Take for example Figure 2, where two processes inter-operate in a nested manner to satisfy a Container Notification requirement. In this process, the Dispatch Officer of a Courier Organization sends a manifest to a SubContracting Courier to signify the arrival of a number of containers on a particular airline. The SubContracting Courier uses the information contained within the manifest to allocate processing resources and to confirm their take-on of the containers. The Send Manifest activity is accumulated with the effect E_1 , and the Receive Manifest activity is accumulated with E_2 , both described below.

- $E_1 \models Knows(DOfficer, Manifest, ClearanceStatus, \mathbf{False})$
- $E_2 \models Knows(POfficer, Manifest, ClearanceStatus, \mathbf{True})$

In this scenario, the analyst has described in their annotation that the manifest to be forwarded has not been allocated a clearance status. This means that there may be a possibility that certain containers will not be present during their subsequent delivery given some regulatory requirements. The annotation to e_2 however, requires that the containers contained in the manifest have been allocated a clearance status to ensure that all expected containers are actually for delivery (i.e. non-cleared containers may not be processed and delivered). The ramification of this error may be the allocation of redundant resources and/or mistaken delivery

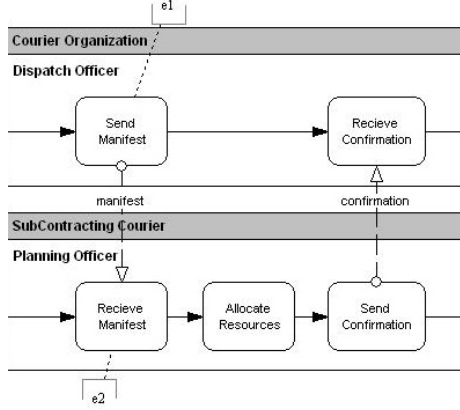


Figure 2. A nested Container Notification process.

of non-cleared containers. Even in the subtlety of this example, an inconsistency has been detected that may have significant ramifications during process deployment and execution.

3.2. Parallel Process Interoperation

Figure 3 outlines a parallel inter-operation between a Courier Organization and a Regulatory Authority to route and screen packages prior to delivery. The process in Figure 3 is in fact incorrect (as will be discussed). In this example the Courier Organization Scans, Assesses, and Routes packages through to their routing destinations via a conveyor belt. The Regulatory Authority Screens, Identifies and may Label packages as they proceed across the conveyor belt. The model has been annotated and the immediate effects of some activities are outlined below.

- $e_1 = \exists s : Status$
 $Knows(SortOfficer, Package, Status, s)$
- $e_2 = \exists s : Status$
 $Knows(SortOfficer, Package, Location, Conveyor) \wedge$
 $Knows(SortOfficer, Package, Status, s) \wedge$
 $Knows(SortOfficer, Conveyor, Type, s) \wedge$
 $Knows(SortOfficer, Conveyor, Mode, \mathbf{Running})$
- $e_3 = \exists s : Status$
 $Knows(SortOfficer, Package, Status, s)$
- $e_4 = Knows(RegulatoryAgent, Package, Status, \mathbf{Held}) \wedge Has(RegulatoryAgent, Package)$
- $e_5 = \exists s : Status$
 $\neg Has(RegulatoryAgent, Package) \wedge$
 $Knows(RegulatoryAgent, Package, Location,$

$Conveyor) \wedge$
 $Knows(RegulatoryAgent, Conveyor, Mode, \mathbf{Running}) \wedge$
 $Knows(RegulatoryAgent, Package, Status, s) \wedge$
 $Knows(SortOfficer, Package, Status, s)$

In addition, the following rule is part of the Regulatory Authority's domain knowledge.

- $R_1 = \forall a_1, a_2 : Agent,$
 $Has(a_1, Package) \Rightarrow$
 $\neg Knows(a_2, Conveyor, Mode, \mathbf{Running})$

This domain specific rule states that if an agent has a package, the agent and any other agent must not know that the conveyor the package is on is in mode 'Running'.

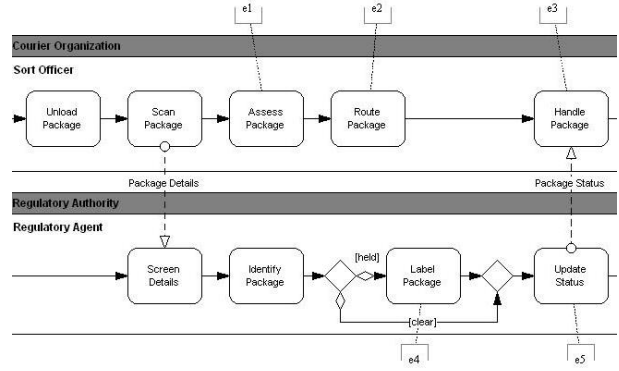


Figure 3. A parallel Package Routing process.

Now, we apply the aforementioned accumulation procedure and determine the following cumulative effects:

- $E_2 = E_3 = \exists s : Status$
 $Knows(SortOfficer, Package, Location, Conveyor) \wedge$
 $Knows(SortOfficer, Package, Status, s) \wedge$
 $Knows(SortOfficer, Conveyor, Type, s) \wedge$
 $Knows(SortOfficer, Conveyor, Mode, \mathbf{Running})$
- $E_4 = Knows(RegulatoryAgent, Package, Status, \mathbf{Held}) \wedge Has(RegulatoryAgent, Package)$
- $E_5^1 = Knows(RegulatoryAgent, Package, Status, \mathbf{Held}) \wedge \neg Has(RegulatoryAgent, Package) \wedge$
 $\exists s : Status$
 $Knows(RegulatoryAgent, Package, Location, Conveyor) \wedge$
 $Knows(RegulatoryAgent, Conveyor, Mode, \mathbf{Running}) \wedge$
 $Knows(RegulatoryAgent, Package, Status, s) \wedge$
 $Knows(SortOfficer, Package, Status, s)$

We only describe one of the effect scenarios in this example - the other would have been determined from the sequential accumulation of effects from ‘Identify Package’.

Now, at the $e_5 \rightarrow e_3$ synchronization we can determine that no inconsistencies exist between the given scenarios using our aforementioned criteria. However, given that certain activities may interleave during parallel interoperation, we also need to test for conflicts that may exist between parallel interoperating activities. If a conflict were to exist between such activities, an effect that is required for some subsequent activity, or even the process in general, may be undone. Therefore, we pairwise compare the cumulative effect scenarios of activities that may interleave to identify any possible conflicts. In the example above, we can detect an inconsistency between the cumulative effect of ‘Route Package’:

- $E_2 \models \text{Knows}(\text{SortOf ficer}, \text{Conveyor}, \text{Mode}, \mathbf{\text{Running}})$,

and ‘Label Package’:

- $E_4 \cup R_1 \models \forall a : \text{Agent} \neg \text{Knows}(a, \text{Conveyor}, \text{Mode}, \mathbf{\text{Running}})$.

Therefore, the interoperation in this case should be deemed inconsistent. The inconsistency has highlighted the violation of a rule requiring a shared object (the *Conveyor*) to not be in mode **Running** if an agent has a package. If this constraint were to be violated at some time, the safety of an agent may be at risk. Given this inconsistency, a change would need to occur to either the process design or implementation of the activities to remove the possibility of this event.

4. Conclusion

In this paper, we have provided a simple and practical approach to support the verification of interoperational business process models. The outcome of applying this verification technique to current business process design and analysis has been illustrated. In order to progress from the current state, we are pursuing tool support. Ideally this will utilize theorem proving technology (e.g. SAT [19]) to provide automated assistance during the propagation of effects and analysis of consistency. The capability to propose minimal alteration to a process model that introduces consistency is also desirable.

References

[1] J. Becker, M. Indulska, M. Rosemann, and P. Green. Do process modelling techniques get better? a comparative ontological analysis of bpmn. In B. C. J. Underwood and

D. Bunker, editors, *Proceedings 16th Australasian Conference on Information Systems*, Sydney, Australia, 2005.

[2] S. Easterbrook, A. Finkelstein, J. Kramer, and B. Nuseibeh. Co-ordinating distributed viewpoints: the anatomy of a consistency check. Technical Report 94/7, Department of Computing, Imperial College, London, 1994.

[3] D. A. Fisher. An emergent perspective on interoperation in systems of systems. Technical Report CMU/SEI-2006-TR-003, Carnegie Mellon Software Engineering Institute, March 2006.

[4] N. E. Fuchs, U. Schwertel, and R. Schwitter. *Attempto Controlled English (ACE), Language Manual, Version 2.0*. Institut für Informatik, Universität Zürich, 1998.

[5] G. Grossman, M. Schrefl, and M. Stumptner. Verification of business process integration operations. In *Proc. 4th International Conference on Business Process Management*, 2006.

[6] V. Gruhn and R. Laue. Using timed model checking for verifying workflows. In J. Cordeiro and J. Filipe, editors, *Proceedings of the 2nd Workshop on Computer Supported Activity Coordination*, pages 75–88. Insticc Press, 2005.

[7] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming* 8, pages 231–274, 1987.

[8] D. Hollingsworth. The workflow reference model version 1.1. Technical Report TC00-1003, Workflow Management Coalition, January 1995.

[9] W. Janssen, R. Mateescu, S. Mauw, and J. Springintveld. Verifying business processes using spin. In E. N. G. Holzman and A. Serhrouchni, editors, *Proceedings of the 4th International SPIN Workshop*, pages 21–36, Paris, France, Nov 1998.

[10] W. Janssen, R. Mateescu, S. Mauw, P. van der Fennema, and P. Stappen. Model checking for managers. In *5th and 6th International SPIN Workshops*, page 92107, 1999.

[11] J. Koehler, G. Tirenni, and S. Kumaran. From business process model to consistent implementation: A case for formal verification methods. In *Proc. 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 96–106, 2002.

[12] A. Krishna, A. Ghose, and A. Vranesovic. Agent-oriented conceptual models to uml sequence diagrams via effect annotations. *International Journal of Multi-Agent and Grid Systems*, Special Issue on Agent-Oriented Software Development Methodologies, 2006.

[13] L. Kutvonen. Addressing interoperability issues in business process management. In *2nd Interop Workshop at EDOC2005*, number B-2005-5 in B-Series. Dept. Computer Science, University of Helsinki, September 2005.

[14] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting bpm processes. In *Proc. 4th International Conference on Business Process Management*, 2006.

[15] L. T. Ly, S. Rinderle, and P. Dadam. Correctness in adaptive process management systems. In *Proc. 4th International Conference on Business Process Management*, 2006.

[16] T. Mitamura and E. H. Nyberg. Controlled english for knowledge-based mt: Experience with the kant system. In *Proceedings of the 6th International Conference on Theoretical and Methodological Issues in Machine Translation*, Leuven, Belgium, July, 5-7 1995.

- [17] A. Paepcke, C.-C. K. Chang, H. Garcia-Molina, and T. Winograd. Interoperability for digital libraries worldwide. *Communications of the ACM*, 41(4):33–43, April 1998.
- [18] D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 63(2):293–304, 1986.
- [19] H. Samulowitz and F. Bacchus. Using sat in qbf. In *International Conference on Principles and Practice of Constraint Programming (CP-2005)*, pages 578–592, 2005.
- [20] R. Schwitter and N. Fuchs. Attempto - from specifications in controlled natural language towards executable specifications. EMISA Workshop ‘Naturlichsprachlicher Entwurf von Informationssystemen’, May, 28-30 1996.
- [21] J. F. Sowa. Common logic controlled english (draft). Technical report, <http://www.jfsowa.com/clce/specs.htm>, February 2004.
- [22] W. van der Aalst. Verification of workflow nets. *Lecture Notes in Computer Science*, 1248:407426, 1997.
- [23] K. van Hee, N. Sidorova, L. Somers, and M. Voorhoeve. Consistency in model integration. *Data Knowl. Eng.*, 56(1):4–22, 2006.
- [24] A. van Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998.
- [25] F. Weitzl and B. Freitag. Checking semantic integrity constraints on integrated web documents. In *ER (Workshops)*, pages 198–209, 2004.
- [26] S. White. Business process modeling notation (bpmn).. Technical report, OMG Final Adopted Specification 1.0 (<http://www.bpmn.org>), February 2006.
- [27] X. Zhao and C. Liu. Tracking over collaborative business processes. In *Proc. 4th International Conference on Business Process Management*, 2006.