Propagation of Event Content Modification in Business Processes

John Wondoh(✉), Georg Grossmann, and Markus Stumptner

University of South Australia, Adelaide, Australia
john.wondoh@mymail.unisa.edu.au, {georg.grossmann,mst}@cs.unisa.edu.au

Abstract. Business processes are composed mainly of activities and events. The latter has gained much focus recently which has resulted in the drift towards *Event-Driven Business Process Management* (EDBPM). Events are used in both monitoring and controlling the execution of business processes. They are considered to be instantaneous and their content cannot be modified after they occur. However, this is not always the case in the real world. An event's content can be modified at runtime under circumstances such as: earlier event information containing errors, or new information being obtained about the event. In such cases, the content modification for that event must be taken into consideration in the execution of the process. Additionally, the modified event's content may affect other events within the process resulting in altering the content of those events as well. Therefore, it is important to determine the propagation of event content modification in an event network within a business process. In this work, we determine the types of event content modifications that can occur within processes, how content modification of one event affects other events within the process, and how the modification affects the process as a whole.

Keywords: Business processes · Event content modification (ECM) · Modification propagation

1 Introduction

Events are important in business process management (BPM) as they provide flexible and effective means of monitoring and controlling the execution of business processes. Events and activities are the main components of business processes with the focus drifting from activities to events in recent years [1]. In fact, the adoption of event processing techniques into BPM has resulted in rigorous research in areas including [9]: event-driven business activity monitoring (EDBAM) [12,17], event-driven process control (EDPC) [8,14], and event-driven predictive analysis (EDPA) [11,20]. In general, all these fall under a single broad area termed event-driven business process management (EDBPM). However, approaches developed in these areas do not consider the possibility of modifying an event's content (attributes), although this occurs in the real world [6,19]. In this work, we develop an approach to improve monitoring and control of

Q.Z. Sheng et al. (Eds.): ICSOC 2016, LNCS 9936, pp. 70–84, 2016.
DOI: 10.1007/978-3-319-46295-0_5

activities within business processes by taking into consideration the possibility of event content modification (ECM).

An ECM may occur as a result of two factors: (i) new information about an event is obtained [6,19], or (ii) errors are detected within the event's information that require correction [16]. In both cases, the content of the original event must be modified to correctly represent the real world or information system. Consider the following example:

Example 1. Some patients discharged from a hospital may require home care services. This responsibility is passed on to a homecare organisation. To provide the services on time, the hospital must assess the patient's health condition and inform the homecare organisation of the planned discharge time of the patient. Based on this information, the homecare organisation will plan scheduled home care visits to the patient to provide the required services. In an EDBPM system, interactions between the homecare organisation and the hospital are driven by events. Figure 1 illustrates a simplified version of the homecare process interacting with a hospital process (not shown) via a service request event. However, this may not happen as planned due to changes that may occur in the business environment. For example, the planned discharge time of the patient may be modified to an earlier time than expected. This leads to the modification of the content of the earlier service request event. Once the ECM becomes known by the homecare organisation, they must take into consideration the modified event's content in their process.

The homecare process consist of several events with a complex network and dependency. To be able to handle an ECM on the fly, two main problems need to be solved:

1. Identifying event dependency within a business process as well as identifying the relationships between attributes of different events. We use the term *event attribute dependency* to describe the latter.
2. Identifying the impact of the ECM of one event on the content of other events within the business process with the aim of identifying how much the business process has been affected by one ECM. For example, modifying the discharge date of a patient should lead to modifying the service delivery date. This is done by utilising identified event dependencies and event attribute dependencies.

Currently, these problems remind unsolved as most event driven approaches in BPM consider either events to be immutable [2,18], or ECM to be limited to a single event [6,16]. In our approach, we capture an event's content modification, and propagate it to other events by utilising event attribute dependencies. We adopt event processing networks (EPNs) [5] for realising event dependencies with the aim of developing an event dependency graph for a given process. The purpose of the event dependency graph is for propagating ECM within the event network. This is important as it helps in identifying how the process can be altered to accommodate certain changes. The main contributions of our work are as follows:

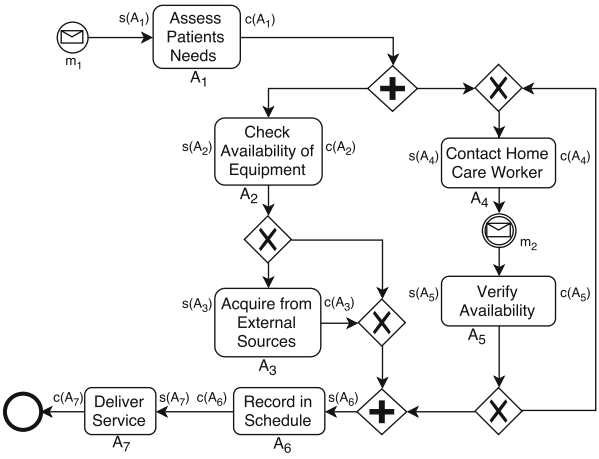

Fig. 1. A homecare process [19]

1. We introduce and formalise the concept of event content modification and event attribute dependencies.
2. We introduce an event dependency graph capable of capturing both event dependencies and event attribute dependencies within business processes.
3. We introduce propagation of event content modification in a business process instance based on our event dependency graph.

The remainder of this paper is organised as follows: Sect. 2 provides a general discussion of events in business processes, followed by Sect. 3 which deals with different types of event dependencies in business processes. Sections 4 and 5 covers event content modification and its propagation. A brief discussion of the usability of the presented approach is given in Sect. 6, followed by a review of related work in Sect. 7. We conclude the paper and discuss future work in Sect. 8.

2 Events in Business Processes

An event is basically a happening within a domain and it may be instantaneous or have a duration. There are three main types of events in business processes [3]: *activity invocation* or *start event* which initiates an instance of an activity type, *activity termination* or *end event* which terminates the execution of an activity instance, and *message* or *request event*. The latter can further be classified as either a *send event* or a *receive event* [7] with respect to the system under consideration, where a send event represents an outgoing message and the receive event represents an incoming message.

The properties of an event include [13] an *event name* (a name describing the type of event), and a *timestamp* (indicating when the event occurred). These properties may vary based on the type of event under consideration as shown

in Table 1. In this table, the column S/H represents a simpler representation of an event. The attributes of an event may be categorised as either mutable (attributes that can be changed) or immutable (attributes that must remain constant). From Table 1, immutable properties of events include: event identification number (id) and activity name. Immutable properties are utilised in event identification. On the other hand, mutable properties include time, additional information (addInfo), role, and payload.

Table 1. Event Structure

Event type	Attributes	Structure	S/H
start event	id, activityName, role, addInfo, time	start{id, activityName, role, addInfo, time}	$s(A)$
end event	id, activityName, role, addInfo, time	end{ id, activityName, role, addInfo, time }	$c(A)$
send event	id, payload, time	send{ id, payload, time }	m
send event	id, payload, time	receive{ id, payload, time }	m

The time attribute captures the time at which an event occurs. Other important temporal attributes not represented in this table include, the detection time of an event, i.e., the time an event becomes known in a system, and its processing time. The latter captures the time an event is utilised in a process, which is application dependent. Hence, it is not captured in the event structure, but is known internally within the business process. The relationship between occurrence time ot_e, detection time dt_e, and processing time pt_e of an event e is such that, $ot_e \leq dt_e \leq pt_e$, except for the case of a future (planned) event where its occurrence is in the future. Temporal constraints of a business process can be expressed in terms of events, and this can prove to be useful in monitoring business processes for constraint violation [4,10,12]. For instance, the start and end events of an activity can be used to determine the duration of the activity. Similarly, the end event of an activity and the start event of a succeeding activity can be used to determine the delay between the two activities.

In Table 1, the payload attribute of a message event represents the information that can be used by the business process in decision making. The addInfo attribute of the start and end events also captures information that will be useful in the execution of the activity. For example, it may provide contextual information required to improve on the quality of the resulting service.

Example 2. Figure 1 shows the different types of events in the process described in Example 1. The start event of an activity is represented by $s(A_i)$ and its end event is represented by $c(A_i)$. Other attributes of each event are ignored for simplicity. 'Receive events', m_1 and m_2, are shown as well. 'Send events' are ignored as they have an external effect but not necessarily on the process.

In the above example, event attributes may be modified after they occur, leading to a possible alteration of the effect of the event. Since the business process interacts with partners such as the hospital, patient, and nurse[1], changes that occur from these systems propagate to the home care process. An example is the change in the proposed discharge date by the hospital (contained in the message event m_1). Other changes may include: nurses changing their availability, and the service requirement of a patient changing. These modifications need to be incorporated into the business process and its impact evaluated.

In the next section we shall discuss event processing networks and how they capture the relationship between events. This is a first step in capturing dependency between events and propagating modification of an event's content to other events within the event network.

3 Event Dependencies in Business Processes

The main concept utilised here is the event processing network (EPN) which captures the relationship between event producers, consumers, event processing agents, and channels. The definition of an EPN as presented in [15] is given as follows:

Definition 1 (Event Processing Network). *An EPN is a graph* $G = (V, E)$ *where* $V = C \cup P \cup A \cup EC$. *V is a set of nodes of four types, with C denoting an event consumer, P denoting an event producer, A denoting an EPA, and EC denoting an event channel.*

$E = \{(u, v) | (u \in (P \vee A) \rightarrow v \in EC) \wedge (u \in EC \rightarrow v \in (C \vee A))\}$,

where E is a set of ordered pairs of nodes representing directed edges. These edges are either between an event producer and an event channel, an event channel and an event consumer, an EPA and an event channel, or an event channel and an EPA.

An event producer introduces an instance of an event into a business process, whereas an event consumer receives and utilises an event [5]. Business activities are first class event producers and consumers as they utilise events for instance activation and produce an event when completed (or cancelled). Other event producers and consumers may be external to the system. A receive event is produced externally and utilised by the process. Similarly, a send event is produced internal but utilised by an external system.

An event processing agent (EPA) processes a set of input events and produces a set of output events [5,15]. EPAs are different from event producers and consumers as they only perform computations on the event without necessarily consuming or producing them. In this work, we limit ourselves to two types of EPAs, compose EPAs which composes a new event from at least two different events, and translate EPAs which translates an event of one type into an event

[1] The nurse may be an external partner, i.e., a firm that outsources to a casual nurse or an internal partner (an employed worker).

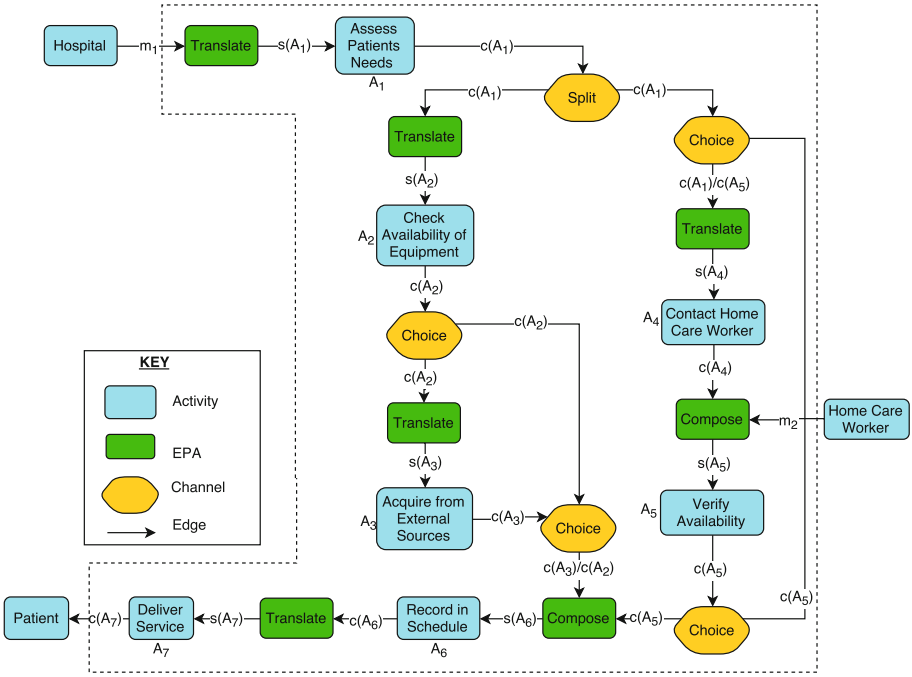

Fig. 2. EPN representation of a homecare process

of another type. An event channel, unlike an EPA, receives a set of input events, makes routing decisions, and sends out the input events unchanged to one or more target processing elements in accordance with these routing decisions [5]. In an EPN representation of a business process, the event channels replace the control flow elements such as the BPMN gateways.

Figure 2 shows the EPN corresponding to the business process of Fig. 1. We have two main types of event producers here, activities within the process as well as external systems that send events to the business process. Note that a key of the various EPN components is provided in Fig. 2.

3.1 Event Dependencies

As shown in Fig. 2, an EPN representation of a business process puts the focus on the steps and components involved in processing events. It is therefore capable of capturing dependencies among events in a process. For instance, it is possible to identify that the occurrence of one event relies on the occurrence of another event. These dependencies can be categorised as either *in/out dependencies* or *temporal dependencies* as defined below.

Definition 2 (In/Out Dependency). *Let A be an EPA or an activity, and let E_i be a set of input events and E_j be a set of output events to A such that*

$E_i \neq \emptyset$, $E_j \neq \emptyset$, $E_i \cap E_j = \emptyset$. We say E_j is dependent on E_i, denoted as $E_i \rightarrow E_j$. The statement $e_i \rightarrow e_j$ is true of events $e_i \in E_i$ and $e_j \in E_j$ if e_i must occur for e_j to occur, i.e., for e_j to be produced by A, e_i must be consumed by A.

Definition 3 (Temporal Dependency). *There exists a temporal dependency between two events e_i and e_j if the occurrence time of e_j, denoted as ot_{e_j}, is dependent on the occurrence time ot_{e_i} of e_i, i.e., $ot_{e_i} \rightarrow ot_{e_j}$. Temporal dependencies may include: $(ot_{e_j} = ot_{e_i}) \vee (ot_{e_j} > ot_{e_i}) \vee (ot_{e_j} < ot_{e_i}) \vee (ot_{e_j} \leq ot_{e_i}) \vee (ot_{e_j} \geq ot_{e_i})$.*

Both types of dependencies are similar and are therefore best utilised together. For instance, an in/out dependency between e_1 and e_2 may have a temporal dependencies of $ot_{e_2} > ot_{e_1}$. While temporal dependencies deal solely with the time of occurrences of events, they do not take into consideration EPAs and activities. Therefore, temporal dependencies facilitate identifying the general ordering of events while in/out dependencies identify the relationship between input and output events.

Utilising Definitions 2 and 3, we can identify dependencies between events. We can further utilise these identified dependencies to realise other types of dependencies within the business process. We break this down into direct and indirect dependencies.

1. Direct Dependencies: In this type of dependency, one event's occurrence is a direct result of another event occurrence. Given two events e_j and e_i, e_j is directly dependent on e_i if (i) $ot_{e_i} \rightarrow ot_{e_j}$ and (ii) $e_i \rightarrow e_j$. For instance, in Fig. 2, $s(A_1)$ is directly dependent on m_1 since m_1 occurs before $s(A_1)$ and $s(A_1)$ is derived from m_1 by translation.
2. Indirect Dependencies: These capture transitive dependencies between events. For instance, in Fig. 2, $s(A_2)$ is directly dependant on $c(A_1)$, while $c(A_2)$ directly depends on $s(A_2)$, therefore, $c(A_2)$ indirectly depends on $c(A_1)$.

Identifying event dependencies are a necessary step to identifying event attribute dependency. This is because attribute dependencies only exist within events that have some form of dependency. Thus, attribute dependency does not exist between unrelated events. We now proceed to discuss event attribute dependencies.

3.2 Event Attribute Dependency

An event's attributes, also referred to as the event's content, may be derived from attributes of other events. The value of these attributes is a function of the other event's attributes. We use the term event attribute dependency to describe some relationship that exists between attributes of different events. A formal definition is given as follows:

Definition 4 (Event Attribute Dependency). *Let a_i and a_j be attributes of event e_i and e_j such that $a_i \in e_i$ and $a_j \in e_j$. a_j is dependent on a_i if*

- *there exists a dependency between e_i and e_j such that $e_i \rightarrow e_j$*
- *there exists a relationship r such that a_j is a function of a_i, i.e., $e_j.a_j = r(e_i.a_i)$.*

If $e_i = e_j$, then the attributes under consideration are attributes of the same event; that is, a self attribute dependency exists within the event. However, an inter-event attribute dependency results when we consider the attribute relationship of two different events, i.e., $e_i \neq e_j$.

Temporal dependency (Definition 3) discussed in Sect. 3.1 is a type of attribute dependency as it compares the occurrence time of events. However, we focus on attributes that are functions of other attributes here, such that modifying one attribute results in the modification of the other. Therefore, we utilise temporal dependencies here to capture general ordering of events relative to each other. In this way, we can capture the relative positions of events in our proposed event dependency graph.

Determining event attribute dependency is important since it aids in determining the attributes of other events that are prone to modification due to an attribute modification of an event. We now proceed to discuss event dependency graphs where we capture both event and event attributes dependencies.

3.3 Event Dependency Graphs

An EPN is important for capturing the relationship between different components in processing an event, i.e., producers, consumers, EPAs and channels. While general event dependencies can be in such a network, the focus is not on capturing event dependencies as well as event attribute dependencies. We propose the use of an event dependency graph (EDG) for this purpose. A dedicated EDG puts the focus on event dependencies as well as capturing inter-event attribute dependency. An EDG is defined as follows:

Definition 5 (Event Dependency Graph). *An EDG is a graph $D = (E, R)$ where $E \neq \emptyset$ is a set of nodes representing events such that $e_i \in E$, and $R \neq \emptyset$ is a set of directed arcs $R \subseteq (E \times E)$ representing dependencies. We distinguish between event dependencies R_e and event attribute dependencies R_a such that for any given EDG, $R_e \cup R_a \in R$ and $R_e \cap R_a \neq \emptyset$.*

Event attribute dependencies (R_a) are represented in an EDG by dashed lines with labels representing the dependency relationship between the two events under consideration. On the other hand, R_e (event dependencies) is represented by solid lines with optional labels representing the type of relationship.

Example 3. Consider the example of an EDG shown in Fig. 3. Event dependencies are captured from m_1 to $c(A_7)$. The attribute discharge date is part of the payload of m_1, while the attribute delivery date of $s(A_7)$ is a function

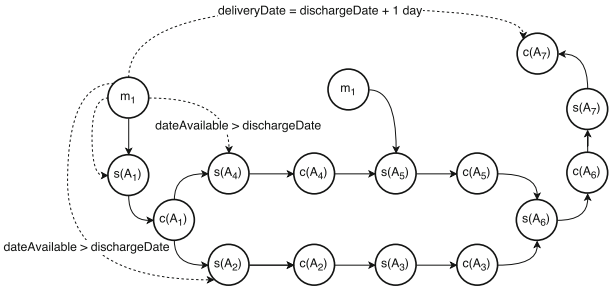

Fig. 3. Event dependency graph for a homecare process

of discharge date, expressed as: $deliveryDate = dischargeDate + 1$, i.e., service delivery should begin a day after the patient has been discharged. These attribute relationships are captured by labels on the arcs. The EDG captures all related events within a particular process and determines dependencies between the attributes of the events. This forms the basics for the propagations of content modification of events. We deal with the latter in the following section.

4 Event Content Modification

Mutable attributes of events are prone to modification or revision, which leads to what we term event content modification. The latter can happen after an event has occurred, has been detected or has been processed. Other literature refers to ECM as event information update [6,19] or revision [16]. Event modification is essential for ensuring that event instances correspond correctly to the real world. Meaning, an event should be able to adapt to changes. The main reasons for ECM is correcting erroneous event information and introducing new event content (resulting from changes in the real world or information system).

There are two main approaches to modifying the content of an event as shown in Fig. 4: (a) utilising a *modification event* e_m that revises the content of the original event, (b) directly updating the event by considering changes in a database. We prefer the modification event approach as it avoids the direct reliance on database systems which may result in additional complexities. A modification event is an event that occurs to modify the content of an existing event. In both cases, a specialised EPA termed as a modification EPA is utilised in modifying the event which we discuss later in this section.

The modified event is considered to be the same as the original event after modification as given in Definition 6. This is because the immutable properties used in event identification are not modified during the modification process. The immutable properties of an event can also be referred to as the event's key.

Definition 6 (Event Equivalence). *Two events e and e' are considered equivalent if the set of immutable event properties, termed key of e is the same as the set of immutable properties key' of e', i.e., $e \equiv e' \Leftrightarrow key = key'$.*

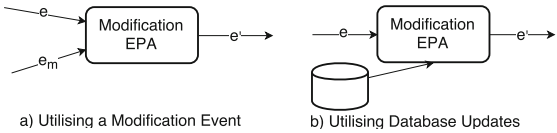

Fig. 4. Event content modification approaches

Since for any given point in time there may exist a set of events with the same key in a system, we utilised the detection time of each event to determine the ordering (Definition 7). This is to enable us determine the latest modified version of the event that should be utilised by the process.

Definition 7 (Event Ordering). *Let e and e' be two events that have occurred in a business process such that $e \equiv e'$ and let dt_e be the detection time of e and $dt_{e'}$ be the detection time of e'. If e' is a modified version of e, then $dt_{e'} > dt_e$.*

Definition 8 (Modification EPA). *Let e and e_m be events within a business process such that e_m is a modification event to e. A modification EPA utilises as inputs e and e_m and produces an output e' such that $e' \equiv e$ and $dt_{e'} > dt_e$.*

A modification EPA may further be categorised into two specialised EPAs based on the type of event modification: *correction EPA* which revise erroneous event attributes with correct information provided by e_m, and *addition EPA* which adds additional content to the existing content of an event.

5 Propagation of Event Content Modification

Modifying the content of one event instance at runtime may affect other events in the business process resulting in ECMs of other events in the process. The notion of propagation of ECM is used here to describe a series of ECMs in an event network resulting from modifying a single event's content. Propagation of ECM is achieved by using the event dependency graph (EDG) and the dependencies that can be derived from it. Propagating modification in an EDG of a business process is essential in ensuring that each event in the network correctly represents the real world. Since the real world is not ideal, changes do occur and these changes must be captured and resolved in the business process. We argue that events being the drivers of processes in an EDBPM should as well be the drivers of change in the process.

In order to determine if an event's content should be modified based on the modification of another event's content, we need to take two important aspects into consideration: event dependency (Definitions 2 and 3) and event attribute dependency (Definition 4). Event dependencies are generic in determining both direct and indirect dependencies between events in the process. Conversely, event attribute dependency is a more specific form of dependency as it specifies dependency between attributes of different events. However, there is a link between

both types of dependency, i.e., event attribute dependency will not exist between attributes of two different events unless one event is dependent on the other. Therefore, to establish attribute dependency, we need to first verify if there exist some generic dependency between the events.

Figure 5 shows a flow chart for identifying events that will be affected by modifying another event's content. It first establishes the need for event dependencies to exist before evaluating for attribute dependency. Event dependencies are either direct or indirect. For direct dependencies, we simply evaluate for attribute dependency between the events. On the other hand, for indirectly dependent events, we need to evaluate for transitive attribute dependency between the events if there exist no direct attribute dependency. In both cases, one of two conclusions are arrived: a) $e_j.a_j$ is modified by modifying $e_i.a_i$, and b) modifying $e_i.a_i$ does not result in the modification of $e_j.a_j$. Making use of such an approach, we can determine all ECMs resulting from a single ECM in a dependency graph for a given business process.

ECM propagation can be represented formally as follows: ECM of event e_i will propagate to event e_j, denoted $e_i' \rightarrow e_j'$ if

1. Direct event and attribute dependency exist: $(a_j \in e_j) \wedge (a_i \in e_i) \wedge (e_j \rightarrow e_i) \wedge (a_j = r(a_i))$.
2. Direct event dependency exist but attribute dependencies must be inferred. We use an event e_k that has an attribute a_k with dependency relationships r_1 with a_i and r_2 with a_j for inference: $(a_j \in e_j) \wedge (a_i \in e_i) \wedge (a_k \in e_k) \wedge (e_k \rightarrow e_j) \wedge (e_i \rightarrow e_k) \wedge (e_i \rightarrow e_j) \wedge (a_j \neq r(a_i)) \wedge (a_k = r_1(a_i)) \wedge (a_j = r_2(a_k))$.
3. Indirect event dependency and direct event attribute dependency exist: $(a_j \in e_j) \wedge (a_i \in e_i) \wedge \neg(e_j \rightarrow e_i) \wedge (e_k \rightarrow e_j) \wedge (e_i \rightarrow e_k) \wedge (a_j = r(a_i))$.
4. Indirect event dependencies and indirect event attribute dependency exist: $(a_j \in e_j) \wedge (a_i \in e_i) \wedge (a_k \in e_k) \wedge \neg(e_i \rightarrow e_j) \wedge (a_j \neq r(a_i)) \wedge (e_k \rightarrow e_j) \wedge (e_i \rightarrow e_k) \wedge (a_k = r_1(a_i)) \wedge (a_j = r_2(a_k))$.

The new value of a_j of event e_j is determined by its relation $r_a \in R_a$ with a_i of event e_i. Given a relation $e_j.a_j = r_a(e_i.a_i)$, modifying the value of a_i must be accompanied by modifying a_j as well. This is to ensure that the attribute relation is not violated. The important aspect of determining a new value of a_j such that the attribute dependency is satisfied is to determine the relation r between a_i and a_j. r is either known during design time and it may be discovered by adopting mining techniques. However, discovering the value r is not in the scope of this paper as we assume that we already have access to this value.

Example 4. Event m_1 which is the message event from the hospital responsible for initiating the homecare process has some attribute dependencies with other events including $s(A_7)$ and $s(A_2)$ in the EDG shown in Fig. 3. The attribute relations are provided in the figure as well: $s(A_7).deliveryDate = m_1.dischargeDate + 1$ (service delivery should begin a day after the patient has been discharged) and $s(A_2).dataAvailable > m_1.dischargeDate$ (equipment must be made available after discharge date). To maintain consistency as well as satisfy the attribute relations, a change in discharge date of m_1 must result in

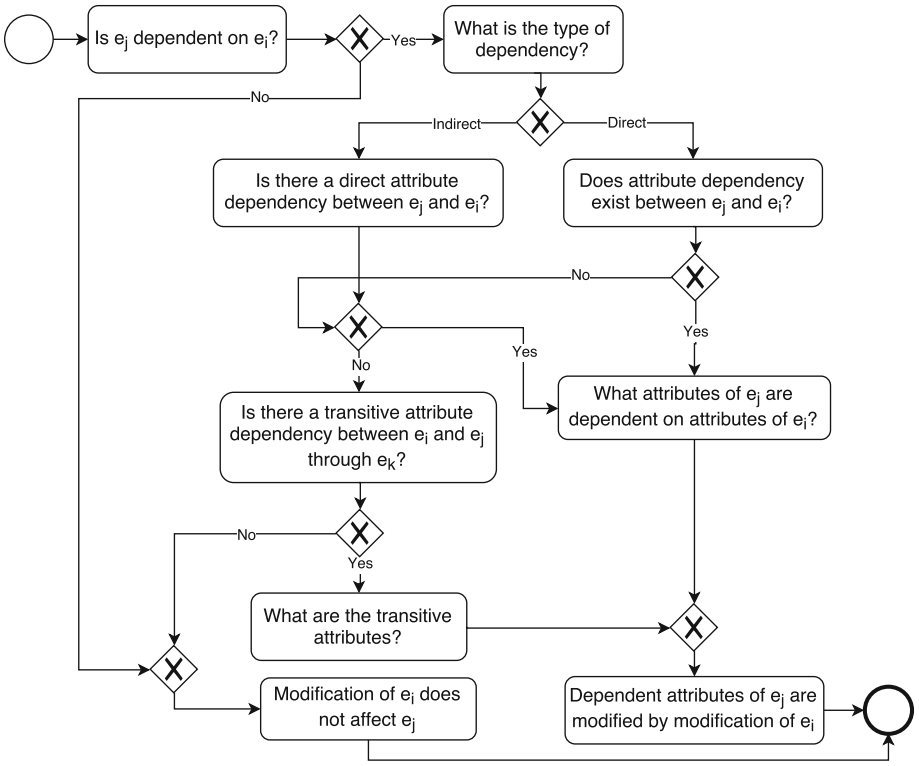

Fig. 5. ECM propagation approach

a change in delivery date of s(A_7), i.e., new deliver date = new discharge date + one day, and a change in availability date of $s(A_2)$ such that new availability data > new discharge date. Using the approach presented in Fig. 5, we can come to this conclusion. First, we realise that there is no direct dependency between m_1 and $s(A_2)$ (or $s(A_7)$); however, a transitive relationship exist between the events. Next we can determine the attribute dependency from the EDG in Fig. 3. Finally, the dependent attributes of $s(A_2)$ and $s(A_7)$ are modified based on the modification of the determinant attribute of m_1.

Identifying and propagating ECM is important as it facilitates process adaptations to changes (especially external changes). The propagation of these changes is driven by events which have become the main drivers of business processes management [1]. The application of the discussed approach in a running business process is given in the discussion in the next section.

6 Discussion

Propagation of ECM in business process presents a less complex approach where the impact of a change in event attributes on a process can easily be determined.

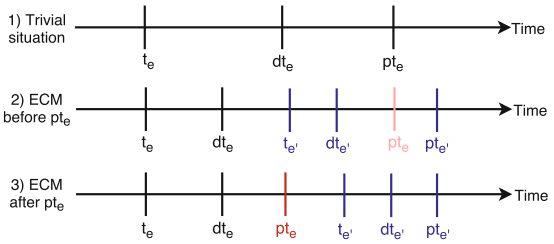

Fig. 6. Processing time relationships

Modification of an event may occur at different times within the lifecycle of a business process or activity. This results in the modified event having different effects on the business process. The processing time of the original event is essential in determining how the modified event affects the business process. Three potential situations may arise when we consider the processing time of an event to the time a modification occurs. Figure 6 shows these three situations. ot_e, dt_e, and pt_e are the occurrence, detection, and processing time of an event e and $ot_{e'}$, $dt_{e'}$, and $pt_{e'}$ are the equivalent time of e' such that $e \equiv e'$.

1. Trivial situation: In this situation the content of the event is not modified, i.e., e' does not occur.
2. ECM before pt_e: In this case, e' occurs and is detected before e is utilised by the process, i.e., $ot_{e'} \leq dt_{e'} < pt_e$. ECM may have no effect on the process instance as the original event's content can be completely ignored.
3. ECM after pt_e: In this case, e' occurs and is detected after the unmodified event is utilised by the process, i.e., $pt_e < ot_{e'} \leq dt_{e'}$. This calls for a modification of the process instance to accommodate for the ECM.

The propagation of ECM improves process adaptability and flexibility by providing information about the potential impact of changes in event attributes. Therefore, an improved approach for handling external contingency can easily be developed. This is beneficial not only to business processes, but also to any event-driven domains, including distributed event-based systems and web services.

7 Related Work

Most work in BPM [1,3,8,12] consider events to be first class citizens and the main driving component of business processes. However, events are only considered to be immutable once they occur, i.e., the event's content cannot be modified after the event has occurred. Therefore, these approaches lean towards the ideal world scenario and are not equipped to handle situations where revision of an events content is required. To handle such a situation, a new event with different identifications are used to represent modification. Therefore, these events must be anticipated during design time and contingency plans put in place to handle

them when they occur. However, this is unachievable in the real world as it is difficult to identify all cases at design time. This is because content modification can happen unexpectedly during runtime of a business process. Determining the impact on the fly by using the approach presented here will ensure improvement in runtime contingency management in business processes.

There is some work outside the domain of business processes that consider event content modification. In Furche et al. [6], they provide an approach for web event information update with the aim of improving decisions that are consistent with the real world. However, they do not consider propagation of the update in an event network as they considered only single composite web events. A similar approach is presented in Sripada [16] where they provide an approach for revising erroneous event information. Similarly, they do not provide an approach to determine other events that may be affected by the change.

In our earlier work [19], we considered propagation of temporal modification of an event in a business process. This approach determines an update to an events temporal attribute and determines its propagation by utilising activity lifecycles and the underlying data model of the process. This approach does not make use of other events that may exist within the business process. Compared to the approach presented in this work (which utilises only events), it is complex and tedious as it considers data model dependencies as well as activity lifecycles. In addition, it focuses only on temporal attributes while ignoring other potentially mutable attributes.

8 Conclusion

In this work, we provide an approach that captures modification in an event's content and propagates this modification to other events within a business process by utilising an event dependency graph. We exploit dependencies that exist between event attributes and we capture these in an event dependency graph. The extent to which a single event's content modification can affect a process at a point in the time can be determined by using this approach. Therefore, a suitable contingency plan can easily be adopted to manage the impact. In our future work, we shall develop and implement an intervention approach for contingency management that utilises the proposed techniques in this work for identifying and determining the impact of contingencies on a business process.

Acknowledgements. This research was partially funded by the Data to Decisions Cooperative Research Centre (D2D CRC).

References

1. Buchmann, A., Appel, S., Freudenreich, T., Frischbier, S., Guerrero, P.E.: From calls to events: architecting future BPM systems. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 17–32. Springer, Heidelberg (2012)
2. Chakravarthy, S., Mishra, D.: Snoop: an expressive event specification language for active databases. Data Knowl. Eng. **14**(1), 1–26 (1994)

3. Damaggio, E., Hull, R., Vaculín, R.: On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. Inf. Syst. **38**(4), 561–584 (2013)
4. Eder, J., Panagos, E., Rabinovich, M.I.: Time constraints in workflow systems. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 286–300. Springer, Heidelberg (1999)
5. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications Company, London (2010)
6. Furche, T., Grasso, G., Huemer, M., Schallhart, C., Schrefl, M.: Bitemporal complex event processing of web event advertisements. In: Lin, X., Manolopoulos, Y., Srivastava, D., Huang, G. (eds.) WISE 2013, Part II. LNCS, vol. 8181, pp. 333–346. Springer, Heidelberg (2013)
7. Knuplesch, D., Reichert, M., Kumar, A.: Visually monitoring multiple perspectives of business process compliance. In: Proceedings of BPM, pp. 263–279 (2015)
8. Koetter, F., Kochanowski, M., Kintz, M.: Leveraging model-driven monitoring for event-driven business process control. In: Proceedings of EMoV, pp. 21–33 (2014)
9. Krumeich, J., Weis, B., Werth, D., Loos, P.: Event-driven business process management: where are we now? A comprehensive synthesis and analysis of literature. Bus. Process Manag. J. **20**(4), 615–633 (2014)
10. Lanz, A., Weber, B., Reichert, M.: Time patterns for process-aware information systems. Requir. Eng. **19**(2), 113–141 (2014)
11. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: Monitoring, prediction and prevention of SLA violations in composite services. In: Proceedings of ICWS, pp. 369–376 (2010)
12. Montali, M., Maggi, F.M., Chesani, F., Mello, P., van der Aalst, W.M.P.: Monitoring business constraints with the event calculus. ACM TIST **5**, 17:1–17:30 (2013)
13. Patri, O.P., Sorathia, V.S., Panangadan, A.V., Prasanna, V.K.: The process-oriented event model (poem): a conceptual model for industrial events. In: Proceedings of DEBS, pp. 154–165. ACM (2014)
14. Scheer, A.W., Thomas, O., Adam, O.: Process modeling using event-driven process chains. In: Process-Aware Information Systems, pp. 119–146 (2005)
15. Sharon, G., Etzion, O.: Event-processing network model and implementation. IBM Syst. J. **47**(2), 321–334 (2008)
16. Sripada, S.M.: A logical framework for temporal deductive databases. In: Proceedings of VLDB, pp. 171–182 (1988)
17. Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 182–198. Springer, Heidelberg (2011)
18. Wieringa, R.J.: Design Methods for Reactive Systems: Yourdon, Statemate, and the UML. Elsevier, Amsterdam (2003)
19. Wondoh, J., Grossmann, G., Stumptner, M.: Utilising bitemporal information for business process contingency management. In: Proceedings of APCCM, pp. 45:1–45:10. ACM (2016)
20. Zeng, L., Lingenfelder, C., Lei, H., Chang, H.: Event-driven quality of service prediction. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 147–161. Springer, Heidelberg (2008)