Feedback on Differences between Business Processes

Remco Dijkman

Eindhoven University of Technology, The Netherlands

Abstract. This paper presents techniques to pinpoint differences between business processes. We say that two processes are different if they are not (completed trace) equivalent. We developed techniques to point out where two processes are different and to explain why they are different. This in contrast to techniques that provide simple true/false answers about whether two processes are equivalent or not. We developed the techniques by first formalizing frequently occurring differences that we discovered in practice and subsequently developing the algorithms that detect these differences. The techniques can be used for various purposes, such as detecting differences between processes in a merger between organizations.

Keywords. Business Process Modelling, Process Equivalence, Analysis

1 Introduction

In many practical cases it is necessary to check whether or not two processes are equivalent. For example, to determine if two organizations van merge their processes without changing them. Also, to determine if an organisation's processes adhere to a standard process [13]. Or, to determine whether a service of an organization is correctly implemented by its internal processes [3].

There exist many techniques to determine whether or not two processes are equivalent. A detailed survey is done by van Glabbeek [18]. However, these techniques provide limited feedback when processes are *not* equivalent, while in most practical cases this feedback is more important than the equivalent/not-equivalent statement. For example, it is unlikely that the processes of two organizations that engage in a merger are equivalent. Therefore, feedback is needed to analyse and resolve the differences. (Of course equivalence checking techniques *can* be adapted to provide useful feedback, which is exactly what we will do in this paper.)

There are two reasons for the fact that equivalence checking techniques provide limited feedback. Firstly, equivalence checking techniques are defined on the execution traces or the state-space of processes, while business processes are defined in terms of activities and relations between those activities. Hence, it is hard to pinpoint exactly *where* those processes are different in terms of the activities and relations. Secondly, when an equivalence checking technique determines that two processes are different, it provides little information as to *why* this may be. If it provides any diagnostic information at all (many state-based techniques return a simple true/false answer), it again returns an answer in terms of states or traces.

However, it is hard to translate this information into feedback that can be used to correct the difference.

For example, consider the simple case in which two processes are different, because an activity exists in one process but not in the other. A trace-based technique would then return a set of traces that can be performed in one process but not in the other. However, a single notification that the activity does not exist could have been returned. Also, it is not possible to distinguish the reason why the processes are different. For example, if the trace [*a b*] can be performed in one process, but not in the other, it is not possible to say whether that is, because in the other process *b* does not exist, or because *a* and *b* are swapped, or because [*a b*] must be followed by *c*, or for some entirely different reason. A state-based technique would return equivalent states from which the processes start deviating. However, if the processes contain any more differences, these differences will not be returned. First one difference must be resolved, and then the algorithm can be run again to check for more differences. Also, as with trace-based techniques, there will be no distinction between this difference and various other differences.

To overcome these difficulties, we developed feedback techniques that present differences between business processes in such a way that each difference:

1. can be precisely pointed out in the business processes; and
2. provides diagnostic information that can be used to resolve the difference.

To meet these requirements, we made a classification of differences between business processes that we obtained from practice. The business processes originated from different departments with similar business functions, such as departments that serve different geographical areas and are geographically separated. We obtained 17 classes of differences [11]. This paper formally defines each class and presents algorithms that can point them out the differences in each class.

Using such a classification helps meet the requirements, because the differences are defined in terms of activities and relations, the same terms in which business processes are defined. Therefore, they can be precisely pointed out in processes. Also, because each difference can be classified into one of 17 different classes, it provides more diagnostic feedback to the user about why two processes are different. For example, because an activity does not exist in one process, because there is a loop in one process but not in the other, because there is a relation between two activities in one process but not in the other, etc.

This paper focuses on differences between business processes with respect to activities and their relations. Also, this paper uses a completed trace semantics and can only detect differences that can be distinguished by such a semantics. It is known that more distinguishing semantics exist [18].

The remainder of this paper is structured as follows. Section 2 introduces the business process formalism that we use and shows how similarities between business processes can be discovered and specified. Section 3 defines the notion of completed trace semantics, necessary to understand the formalization of process differences. Section 4 formally defines the process differences and the algorithms that can detect them. This constitutes the core contribution of this paper. Section 5 presents a case-study in which we applied our techniques. Section 6 discusses the relation between the differences from section 4 and existing equivalence checking techniques. Section 7 presents related work. Section 8 presents conclusions and future work.

2 Business Processes and Similarities between them

Business processes can be represented in many different notations. However, we want to define our difference detection techniques in a notation-independent manner, to allow for situations in which we want to detect differences between business processes that are defined in different notations (which includes many practical situations). Therefore, this section defines an abstraction of business processes. Different notations can be mapped onto this abstraction, to make the detection techniques work for that notation.

First, this section defines the abstract business process notation. Second it presents the notation that is in the remainder of this paper and that is mapped to the abstract business process notation. Third, it explains how similarities between business processes can be expressed. The definition of similarities between business processes is necessary before differences can be detected. Fourth, it shows an example of two business processes and their similarities. This example will be used as a running example throughout this paper.

2.1 Abstract Business Process Notation

The abstract business process notation used in this paper corresponds to the class of (terminating) standard workflow models [20], claimed to be the 'natural' interpretation of business processes for use in workflow engines. A wide range of tools and techniques exists for this class of business processes.

A business process in the abstract business process notation is defined as follows.

Definition 1 (abstract business process). An abstract business process is defined by a tuple (Σ, C, R), where:

- Σ is the set of activity nodes in the process;
- C is a set of control nodes in the process; and
- $R \subseteq (\Sigma \cup C) \times (\Sigma \cup C)$ is a relation between activities and control nodes.

Note that we do not commit to a specific set of control nodes. We only assume that the control nodes that are used have a precise semantics (see below). Also note that we do not consider a hierarchy of processes and sub-processes. If such a hierarchy must be analysed, the processes in that hierarchy must be 'flattened' into a single process (which is what we did for a hierarchy of processes in the Business Process Modelling Notation [26]).

Definition 2 (input). The input of an activity $a \in \Sigma$ in a process P (denoted $in_P(a)$) is the set consisting of a and the activities that directly precede a. An activity b directly precedes a iff $(a, b) \in R$ or there exist $c_1, \ldots, c_n \in C$, such that $(a, c_1), (c_1, c_2), \ldots, (c_{n-1}, c_n) \in R$.

Definition 3 (abstract business process semantics). The semantics of an abstract business process is defined by a tuple $(S, \Sigma_\tau, \rightarrow, s_0, S_f)$ where:

- S is the finite set of states that the process can be in;
- Σ_τ is the set of activities in the process, and a distinguished element $\tau \in \Sigma$ denoting an activity with no visible effect (i.e. a "silent activity");
- $\rightarrow \subseteq S \times \Sigma_\tau \times S$ is the transition relation of the process and the transition (s, a, s') $\in \rightarrow$ denotes that whenever the process is in a state s, the occurrence of some action a allows the process to transition into a state s' (we assume that all activities $a \neq \tau$ are used in the process);
- $s_0 \in S$ is the single initial state of the process; and
- $S_f \subseteq S$ is the set of final states of the process.

Notation. We write $s_1 \rightarrow^a s_2$ to denote that $(s_1, a, s_2) \in \rightarrow$. Σ^* is the set of traces (i.e. lists) over an alphabet Σ. We write $s \rightarrow^a$ to denote that there exists a state s' such that $(s, a, s') \in \rightarrow$. There is a path of silent transitions from s to s' (denoted $s \Rightarrow s'$) iff $s = s'$ or there exists an s_1 such that $s \rightarrow^\tau s_1 \Rightarrow s'$. We write $s \Rightarrow^a s'$ iff there exist an s_1 and s_2 such that $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$. For a trace $\sigma = a_1\ a_2\ a_3 \ldots a_n$, there is a path from s to s' (denoted $s \Rightarrow^\sigma s'$) iff there exist $s_1, s_2, \ldots$ such that $s \Rightarrow^{a1} s_1 \Rightarrow^{a2} s_2 \Rightarrow^{a3} \ldots \Rightarrow^{an} s'$.

We assume that the business process is *sound* (derived from the soundness criteria for workflow nets [5] and corresponding to the notion of *terminating* standard workflow models [20]), meaning that processes eventually always complete in a final state. More precisely:
- the initial state is a 'source state' (it only has outgoing transitions): $\neg\exists\ s' \in S, a \in \Sigma : (s', a, s_0) \in \rightarrow$;
- all final states are 'sink states' (they only have incoming transitions): $\forall s_f \in S_f: \neg\exists s' \in S, a \in \Sigma : (s_f, a, s') \in \rightarrow$;
- all states are reachable from the initial state: $\forall s \in S: \exists\sigma \in \Sigma^*: s_0 \Rightarrow^\sigma s$; and
- from all states the process can complete in a final state: $\forall s \in S: \exists\sigma \in \Sigma^*, s_f \in S_f: s \Rightarrow^\sigma s_f$.

In this way the notion of abstract business process model is closely related to that of workflow net and terminating workflow net, with the exception that it assumes a finite state-space. However, most processes in practice have a finite state-space.

2.2 Business Processes in the Business Process Modelling Notation

Although our techniques can be used for several concrete process modelling notations, we will use the Business Process Modelling Notation (BPMN) [26] in the remainder of this paper.

Figure 1 shows the elements of the BPMN notation that we use for this paper. There is a single activity node (in terms of the definition of process above) called task, which represents an activity in the process. The relation between activities and control nodes is represented by a flow relation. A flow from one node to another represents that when the flow is enabled, it enables the node that it points to. Upon completion a task enables the flows that leave it. We use the following control nodes.

- Start event. Represents the start of a process. Flows leaving the start event are enabled at the start of the process. No flows can point to it.
- End event. Represents the end of a process. No flows can leave from it.
- Decision gateway. Represents a choice in the process. After it is enabled, it can enable exactly one flow that leaves from it.
- Merge gateway. Represents a merge between several alternatives. It is enabled if one of the flows that point to it is enabled. After it is enabled, it can enable the flow that leaves it.
- Fork gateway. Represents the start of parallel paths in a process. After it is enabled, it enables all flows that leave from it.
- Join gateway. Represents the end of parallel paths in a process. It is enabled if all flows that point to it are enabled. After it is enabled it can enable the flow that leaves from it.

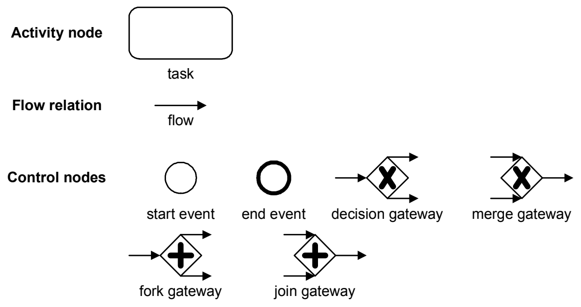


Figure 1. BPMN Notation

In separate work we have defined a semantics of BPMN [12] as well as a tool chain that allows checking whether or not a given BPMN process model is sound and safe. In this paper, we will not discuss the formal semantics of the notation used, but instead we will rely on the reader's intuitive understanding of the semantics of BPMN and refer to our previous work for the technical details of the BPMN semantics.

2.3 Business Process Similarities and Activity Relations

To analyse the differences between business processes, their similarities must first be defined. After all, without an indication as to where the processes correspond, it is impossible to precisely indicate where they are different.

Logical points for indicating similarities between processes are the activities in those processes, because activities are clearly identifiable building blocks, which must be similar for the processes that contain them to be similar. Therefore, we use relations between activities to indicate similarities between processes. However, it is unlikely that activities between processes are equivalent in a 1-to-1 manner and that the names of equivalent activities are identical, such that determining equivalence is simply a matter of comparing names. Instead, the process analyst will have to define how activities are related. To aid him in this task, this section presents some frequently occurring relations between activities, which we discovered in practice [11].

Equivalent activities. Two activities are equivalent for the purpose of the analysis if:
1. the effect that they have on the environment is the same (e.g. the same information is recorded in a database or the same people are informed of a decision); and
2. the way in which they achieve this effect is the same (e.g. the same database is used to record the information and the same means of communication is used to inform people of a decision).

If these two criteria are met we say that the activities represent *the same unit of work*. The requirement that these activities or roles are equivalent *for the purpose of the analysis* is important, because depending on the purpose of the analysis different decisions can be made on equivalence. Moreover, depending on the purpose of the analysis, the second requirement for equivalence may be dropped.

For example, if the purpose of the analysis is to develop a common information system, then the activity of entering personal information of a client in information system 'A' can be considered equivalent to entering this information in information system 'B', because once the common information system is in place, this activity will be exactly the same. However, if the purpose of the analysis is to merge departments, but not information systems, then the activities are different, because they will be different in the merged process; an employee trained on one system cannot use the other without the proper training and data stored in one system will not be available in the other.

Note that the relation can be between any number of activities from the two processes. In particular two activities a_1 and a_2 from one process can be defined to be equivalent to one activity b from another process. In that case the occurrence of either a_1 or a_2 corresponds to the occurrence of b.

Figure 2.i shows an example of equivalent activities.

Subsumed activity. An activity in one process subsumes an activity in another process if it represents the same unit of work as the other activity, but includes another unit of work as well. Figure 2.ii shows an example.

Partly corresponding activities. An activity in one process partly corresponds to an activity in another process, if these activities partly represent equivalent units of work, but both also represent other units of work. Figure 2.iii shows an example.

Interchanged activities. Two activities are interchanged for the purpose of the analysis if:
1. the effect that they have on the environment is the same (equivalence criterion 1 above); but
2. the way in which they achieve this effect is *not* the same (equivalence criterion 2 above).

The distinction between equivalent and interchanged activities is only relevant if both the equivalence criteria explained for the purpose of the analysis, because if only equivalence criterion (1) is used, interchanged activities are equivalent by definition. Figure 2.iv shows an example of interchanged activities.

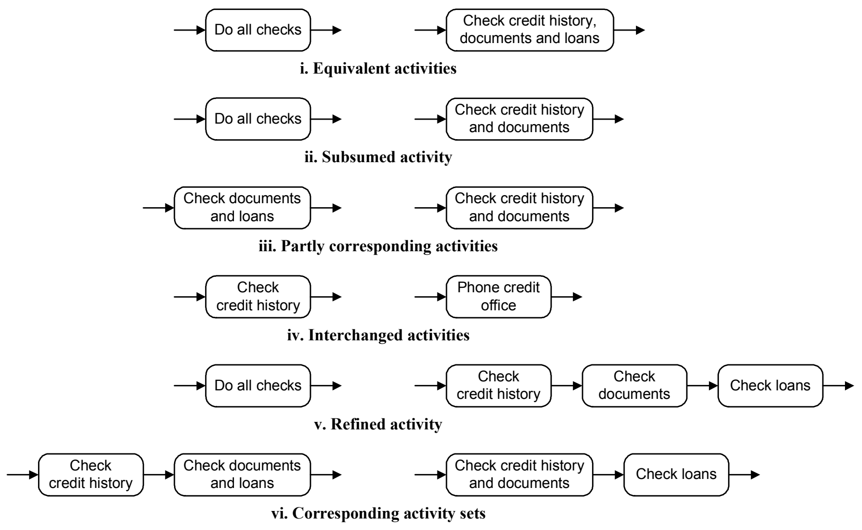

Do all checks
Check credit history, documents and loans
i. Equivalent activities
Do all checks
Check credit history and documents
ii. Subsumed activity
Check documents and loans
Check credit history and documents
iii. Partly corresponding activities
Check credit history
Phone credit office
iv. Interchanged activities
Do all checks
Check credit history
Check documents
Check loans
v. Refined activity
Check credit history
Check documents and loans
Check credit history and documents
Check loans
vi. Corresponding activity sets

Figure 2. Examples of Activity Relations

Refined activity. A set of activities in one process refines a single activity in the other process, if:

1. it corresponds to that activity according to one of the relations above (it is equivalent, subsumes, is subsumed by or partly corresponds to that activity);
2. there is no relation between an activity in the set of activities and another activity in the other process; and
3. the set of activities cannot be changed such that there is a closer correspondence.

Where we consider:

- equivalence to be a closer correspondence than subsumption;
- subsumption to be a closer correspondence than partial correspondence;
- a set of activities that represents more of the unit of work of the single activity to be a closer correspondence than a set that represents less; and
- a set of activities that represents less of a unit of work that is not represented by the single activity to be a closer correspondence than a set that represents more.

Or, put simply, the set of activities should represent as much as possible of what the single activity represents and as little as possible of what the single activity does not represent.

We say that the set of activities *refines* the single activity, because it corresponds to the same unit of work, but describes that unit of work at a higher level of granularity.

Figure 2.v shows an example of refined activities.

Corresponding sets of activities. A set of activities in one process corresponds to a set of activities in another process, if:

1. it corresponds to that set of activities according to one of the relations above (it is equivalent, subsumes, is subsumed by or partly corresponds to that set of activities);

2. there is no relation between an activity from either set and an activity that is not in the other set;
3. it is not possible to split up the sets into multiple corresponding sets, refinements or singular activities; and
4. the sets of activities cannot be changed such that there is a closer correspondence.

Where we consider:

- two sets of activities that represents more of the same unit of work to be a closer correspondence than two sets that represents less; and
- a set of activities that represents less of a unit of work that is not represented by the other to be a closer correspondence to the other than a set that represents more.

Figure 2.vi shows an example of corresponding sets of activities.

Definition 4 (activity relation). An activity relation can be described by a tuple (*r*, *type*), where:

- $r \in \wp(\Sigma) \times \wp(\Sigma)$ describes the relation between sets of activities (or single activities in case a set consists of only one activity); and
- $type \in \{equivalent, partly, subsumes, interchanged\}$ describes what type of relation exists between the sets of activities: an equivalence relation, a partial equivalence relation, a subsumes relation or an interchanged relation.

We say that sets of activities that are related by *r* are *similar*, because we will treat them as equivalent for the purpose of further comparison of the processes.

Now that we established the similarity between the processes through the relation, we can compute the differences between the behaviours that they represent. For the purpose of computing differences between the behaviours we compare the order of occurrences of related (sets of) activities. However, because of the way in which behavioural equivalence is defined, we can only compare the occurrence of single activities, not of sets of activities. Therefore, we require the process analyst to define the completion occurrence relation.

Definition 5 (completion occurrence relation). The completion occurrence relation $\sim \subseteq \Sigma \times \Sigma$ relates activities such that: $(a, b) \in \sim$ represents that the completion of *a* is equivalent to the completion of *b* and if *a* or *b* is in a set, its occurrence also marks the completion of that set. ~ is an equivalence relation and therefore reflexive, symmetric and transitive.

To make life easier for the process analyst, we can add $(a, b) \in \sim$ if $(\{a\}, \{b\}) \in r$, because if two singular activities are defined to be similar, clearly their completion should coincide. It is possible to add the completion occurrence relations for some sets of activities too. For example: in case the set contains a sequence of activities, the completion of the last activity in the sequence corresponds to the completion of the set (Figure 3.i); and in case the set contains an exclusive choice between two or more activities (possibly preceded by a sequence), the completion of either one of the activities in the choice corresponds to the completion of the set (Figure 3.ii).

There are also sets of activities for which it is impossible to specify the completion occurrence relation. These are the sets of activities in which multiple occurrences of the same or different activity represent the completion of the set (because the completion occurrence relation is defined on *any* occurrence of *a single* activity). For example: in case the set contains a fork between two activities, only the completion of both activities corresponds to the completion of the set (Figure 3.iii); in case the set ends with a loop (Figure 3.iv), it is unclear if the occurrence of the last activity corresponds to the completion of the set or another iteration of the loop will be performed. If a process contains such constructs, it is possible to perform a pre-processing step to remove these constructs. In the pre-processing step, stub-activities should be added after the problematic constructs, such that the stub-activities correspond to the completion of multiple other activities. Figure 3.v and Figure 3.vi show how these stub-activities can help solve the problems for the constructs from Figure 3.iii and Figure 3.iv, respectively.

We leave a precise analysis of the completion of a set of activities and the extent to which the completion occurrence relation for a set can be (automatically) derived, for future work.

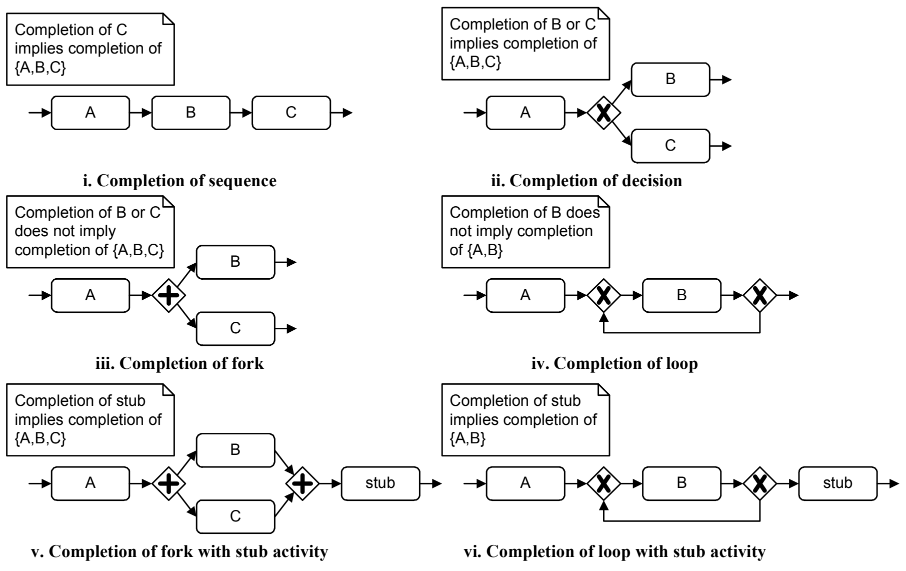

Completion of C implies completion of {A,B,C}
A
B
C
i. Completion of sequence
Completion of B or C implies completion of {A,B,C}
A
B
C
ii. Completion of decision
Completion of B or C does not imply completion of {A,B,C}
A
B
C
iii. Completion of fork
Completion of B does not imply completion of {A,B}
A
B
iv. Completion of loop
Completion of stub implies completion of {A,B,C}
A
B
C
stub
v. Completion of fork with stub activity
Completion of stub implies completion of {A,B}
A
B
stub
vi. Completion of loop with stub activity

Figure 3. Examples of Activity Sets and their Completion

Although the completion occurrence relation is an equivalence relation, in practice a process analyst may define a relation that is not reflexive, symmetric or transitive. Typically, we expect a process analyst to only define the relationships between activities from different processes and not between activities from the same process. However, we rely on the relation to be reflexive, symmetric and transitive in the remainder of this paper. Therefore, if the process analyst defines ~ such that it is not reflexive, symmetric or transitive, we compute the reflexive, symmetric and transitive closure to turn it into an equivalence relation.

2.4 Running Example: Business Processes and their Similarities

Figure 4 shows the running example that we use in the remainder of this paper. The figure shows two business processes for mortgage processing. For the purpose of the example, we used identical names for equivalent activities.

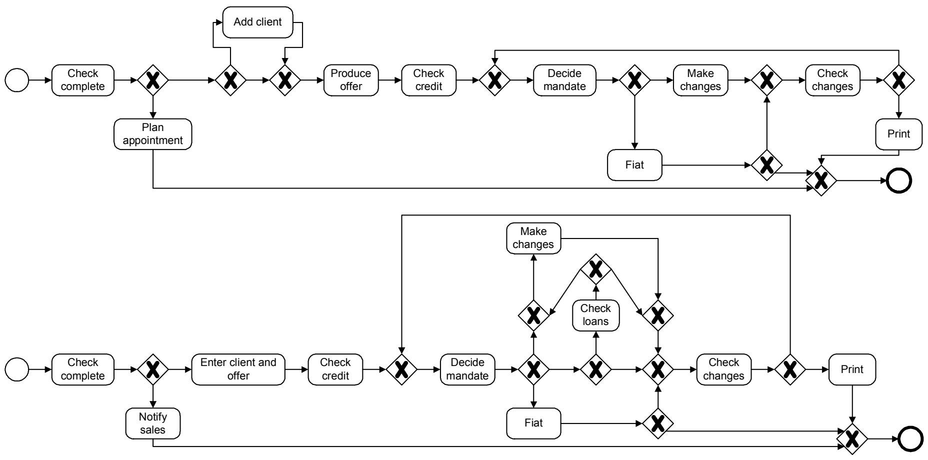


Figure 4. Running Example

In the example equivalent names represent equivalent activities, 'Enter client and offer' refines 'Add client' and 'Produce offer'. 'Notify sales' is interchanged with 'Plan appointment'. So the relations between activities in the processes are defined as:
{ (({'Check complete'},{'Check complete'}), *equivalent*),
(({'Plan appointment'},{'Notify sales'}), *interchanged*),
(({'Enter client and offer'},{'Add client', 'Produce offer'}), *equivalent*),
(({'Check credit'},{'Check credit'}), *equivalent*),
(({'Decide mandate'},{'Decide mandate'}), *equivalent*),
(({'Make changes'},{'Make changes'}), *equivalent*),
(({'Fiat'},{'Fiat'}), *equivalent*),
(({'Check changes'},{'Check changes'}), *equivalent*),
(({'Print'},{'Print'}), *equivalent*)}

We assume that each pair of equivalent or interchanged activities has a completion occurrence relation. We also assume 'Enter client and offer' has a completion occurrence relation with 'Produce offer'. So the equivalence relation (~) is defined as:
{ ({'Check complete'},{'Check complete'}),
({'Plan appointment'},{'Notify sales'}),
({'Enter client and offer'},{'Produce offer'}),
({'Check credit'},{'Check credit'}),
({'Decide mandate'},{'Decide mandate'}),
({'Make changes'},{'Make changes'}),
({'Fiat'},{'Fiat'}),
({'Check changes'},{'Check changes'}),
({'Print'},{'Print'})}

3 Completed Trace Equivalence

The relations between activities from two business processes provide the basis for checking the differences between the organizational behaviour that is represented by the processes. We refer to behavioural differences as *procedural differences*. In this paper we define procedural differences in terms of (in)equivalences between the completed trace-based semantics of processes. We define the notion of completed trace-based semantics and equivalence below.

Definition 6 (completed trace semantics). Given a process $P = (S, \Sigma_\tau, \rightarrow, s_0, S_f)$, the completed trace semantics of P, written $Tr(P)$, is defined as follows: $Tr(P) = \{\sigma \in \Sigma^* \mid s_0 \Rightarrow^\sigma s_f, s_f \in S_f\}$. We denote the trace with length 0 (the empty trace) as ε. In the remainder of this paper we will assume completed trace semantics and use the terms trace semantics and completed trace semantics interchangeably.

Definition 7 (completed trace equivalence). Two processes, P and Q, are defined to be completed trace-equivalent if they have equivalent sets of traces: $Tr(P) = Tr(Q)$. In the remainder of this paper we will use completed trace equivalence and use the terms trace equivalence and completed trace equivalence interchangeably.

In this paper we must consider the defined similarities between activities of P and Q. This means that two activities from P and Q, respectively, are similar if they are defined to be similar by the process analyst. However, the traditional notion of trace equivalence considers activities to be similar if they are 'the same'. Therefore, we must define a notion of trace equivalence that considers similarities as they are defined by the process analyst.

Definition 8 (activity equivalence). Given two processes P and Q, a relation $\sim \subseteq \Sigma_P \times \Sigma_Q$, is an *activity equivalence* iff $\sim$ is reflexive, symmetric and transitive.

Given an activity equivalence relation $\sim \subseteq \Sigma_P \times \Sigma_Q$ between processes P and Q and an activity $a \in \Sigma_P \cup \Sigma_Q$, the set of equivalent activities of a is represented by $[a]_\sim$ (also known as the equivalence class of a for equivalence relation $\sim$).

Definition 9 (trace semantics modulo activity equivalence). Given an activity equivalence relation $\sim$ between processes P and Q, the set of traces of P modulo $\sim$, written $[Tr(\mathrm{P})]_\sim$, is the set of traces obtained by replacing in every trace of $Tr(P)$, every occurrence of an action a by its equivalence class $[a]_\sim$. $[Tr(\mathrm{Q})]_\sim$ is defined similarly.

Definition 10. (trace equivalence modulo activity equivalence) Given an activity equivalence relation $\sim$ between processes P and Q, P and Q are trace-equivalent, written $P =_\sim Q$, iff $[Tr(P)]_\sim = [Tr(Q)]_\sim$

The procedural differences defined below focus on the context of single activities (and their equivalent activities). Therefore, we define an operator to restrict the set of traces to the part of those traces that is defined on a particular set of activities.

Definition 11 (trace restriction). For a set of activities *as*, a trace σ and a set of traces *Trs*:

- σ / as is the trace restriction of σ to *as*, obtained by removing from σ any activities that are *not* in *as*.
- $Trs / as = \{\sigma / as \mid \sigma \in Trs\}$ is trace set restriction.

4 Procedural Differences between Business Processes

This section first precisely defines frequently occurring behavioural differences between business processes (which we call procedural differences), which we discovered in practice [11]. Second, it presents techniques for detecting them in a certain sub-class of business processes. Third, it presents and proves relations between the procedural differences, which helps to return only the most distinguishing difference if multiple differences exist with respect to the same activity. Fourth it applies the techniques to the running example from Figure 4.

4.1 Precise Definitions of Procedural Differences

Definition 12 (skipped activity). A skipped activity is an activity for which there is no similar activity in the other process. More precisely, an activity from a process *P* is a skipped activity in the relation *r* between process *P* and *Q*, if it does not appear in any of the sets in the domain or co-domain of *r*.

The running example (Figure 4) contains one skipped activities: ‘Check loans’.

Different input dependencies. If two similar activities can be enabled by different sets of activities, we say that they have different input dependencies. It is possible (but unlikely) that the different input dependencies are irrelevant. However, they are a strong indication that one or more of the following differences exist:

1. there is a procedural difference with respect to the activities, because the different sets of input dependencies allow the activities to be performed at different times in the execution of the processes;
2. the information that is used for performing the activities is different, because the different sets of input dependencies provide different information;
3. the actors that (are authorized to) perform these activities depend on the work done by different sets of actors.

Definition 13 (different input dependencies). An activity a in process P has different input dependencies than its equivalent in process Q, if there exist input activities that directly precedes it in some trace of process P, but in no trace of process Q, or if there exist input activities that directly precede it in some trace of process Q, but in no trace of process P.

- $[in_P(a)]_\sim$ is the set of input activities of a in P, a itself, and all equivalents of these activities.

- The input dependencies of an activity a in a process P is the set of activities xs, such that $x \in xs$ iff there exists a trace $\sigma \in [Tr(P)]_\sim / [in_P(a)]_\sim$ such that $\sigma = \sigma' \, x \, [a]_\sim \, \sigma''$ (for some σ', σ''). (Note that x is an equivalence class.)

- let xs be the set of all input dependencies of a in P and ys be the set of all input dependencies of a in Q.

- a in P and its equivalent in Q have different input dependencies iff $xs \neq ys$.

Figure 5.i illustrates the concept of different input dependencies. In this figure we assume that B_1 and B_2, and C_1 and C_2 have equivalent occurrence. The figure shows how we can compute the difference in dependencies for the equivalent activities C_1 and C_2.

Additional input dependencies. A special case of different input dependencies is the case in which one activity has additional input dependencies with respect to the other. Like with the different input dependencies this difference may indicate an uninteresting difference, but it can also indicate the more serious underlying differences indicated by the different input dependencies. In addition to that, additional input dependencies can indicate that additional information is used to perform an activity.

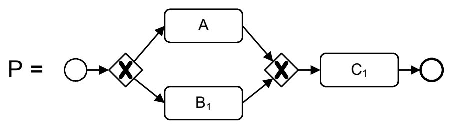
Definition 14 (additional input dependencies). An activity a in process P has additional input dependencies than its equivalent in process Q, if there exists some input activity b that directly precedes it in some trace of process P, but in no trace of process Q.

- for xs and ys defined as in definition 13.

- a in P has additional dependencies with respect to its equivalent in Q iff $xs \supset ys$.

- similarly, a in P has fewer dependencies with respect to its equivalent in Q iff $xs \subset ys$.

Figure 5.ii illustrates the concept of additional input dependencies. In this figure we assume that B_1 and B_2, and C_1 and C_2 have equivalent occurrence. The figure shows how we can compute the additional input dependencies of C_1 with respect to C_2.



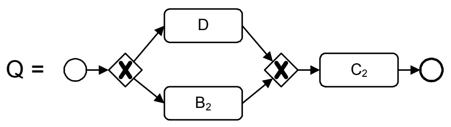
$\mathrm{Tr}(P) = \{A\ C_1, B_1\ C_1\}$

$in_P(C_1) = \{A, B_1, C_1\}$

$[\mathrm{Tr}(P)]_{\sim} = \{\{A\}\{C_1,C_2\}, \{B_1, B_2\}\{C_1,C_2\}\}$

$[\mathrm{Tr}(P)]_{\sim} / [in_P(C_1)]_{\sim} = \{\{A\}\{C_1,C_2\}, \{B_1, B_2\}\{C_1,C_2\}\}$

$xs = \{\{A\}, \{B_1, B_2\}\}$



$\mathrm{Tr}(Q) = \{D\ C_2, B_2\ C_2\}$

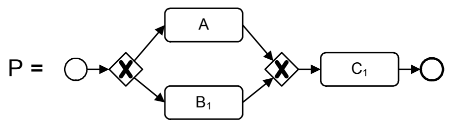
$in_Q(C_2) = \{D, B_2, C_2\}$

$[\mathrm{Tr}(Q)]_{\sim} = \{\{D\}\{C_1,C_2\}, \{B_1, B_2\}\{C_1,C_2\}\}$

$[\mathrm{Tr}(Q)]_{\sim} / [in_Q(C_2)]_{\sim} = \{\{D\}\{C_1,C_2\}, \{B_1, B_2\}\{C_1,C_2\}\}$

$ys = \{\{D\}, \{B_1, B_2\}\}$

i. Different dependencies



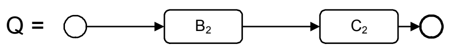
$\mathrm{Tr}(P) = \{A\ C_1, B_1\ C_1\}$

$in_P(C_1) = \{A, B_1, C_1\}$

$[\mathrm{Tr}(P)]_{\sim} = \{\{A\}\{C_1,C_2\}, \{B_1, B_2\}\{C_1,C_2\}\}$

$[\mathrm{Tr}(P)]_{\sim} / [in_P(C_1)]_{\sim} = \{\{A\}\{C_1,C_2\}, \{B_1, B_2\}\{C_1,C_2\}\}$

$xs = \{\{A\}, \{B_1, B_2\}\}$



$\mathrm{Tr}(Q) = \{B_2\ C_2\}$

$in_Q(C_2) = \{B_2, C_2\}$

$[\mathrm{Tr}(Q)]_{\sim} = \{\{B_1, B_2\}\{C_1,C_2\}\}$

$[\mathrm{Tr}(Q)]_{\sim} / [in_Q(C_2)]_{\sim} = \{\{B_1, B_2\}\{C_1,C_2\}\}$

$ys = \{\{B_1, B_2\}\}$

ii. Additional dependencies

Figure 5. Input Dependencies

Different input conditions. If two related activities have different input conditions there is a procedural difference with respect to those activities, meaning that the states in which they can occur differ.

Definition 15 (different input conditions). An activity *a* has different input conditions in *P* than its equivalent in *Q*, if the set of traces, restricted to $in_P(a)$, in *P* is different from the set of traces, restricted to $in_Q(a)$, in *Q*. (See definition 11 for restriction.) We distinguish between *positive input conditions*, which are traces in which *a* occurs, and *negative input conditions*, which are traces in which *a* does not occur.

- If $vs = [Tr(P)]_{\sim} / [in_P(a)]_{\sim}$ is the input condition of *a* in *P* and $ws = [Tr(Q)]_{\sim} / [in_Q(a)]_{\sim}$ is the input condition of *a* in *Q*.

- elements of *vs* and *ws* in which $[a]_{\sim}$ appears are positive input conditions of *a* in *P* and *Q*, respectively, elements in which $[a]_{\sim}$ does not appear are negative input conditions.

- *a* has different input conditions in *P* than its equivalent in *Q* iff $vs \neq ws$.

Figure 6.i illustrates the concept of different input conditions. In this figure we assume that A_1 and A_2, B_1 and B_2, and C_1 and C_2 have equivalent occurrence. The figure shows how we can compute the difference for equivalent activities C_1 and C_2.

Definition 16 (additional input conditions). A special case of a difference between input conditions is when *a* has additional input conditions in one of the processes. If the additional input conditions are only positive input conditions, we say that the input condition is *stricter* in the other process.

- if *vs* and *ws* are the input conditions for *a* in *P* and *Q*, respectively.
- *a* has additional input conditions in *P* than its equivalent in *Q* iff $vs \subset ws$, similarly, *a* has additional input conditions in *Q* iff $vs \supset ws$.
- if $vs \subset ws$ and $ws - vs$ only contains positive input conditions then the input condition for *a* in *P* is stricter, similarly, if $ws \subset vs$ and $vs - ws$ only contains positive input conditions then the input condition for *a* in *Q* is stricter.

Figure 6.ii illustrates the concept of additional input conditions. In this figure we assume that A_1, A_2 and A_3, B_1 and B_2, and C_1 and C_2 have equivalent occurrence. The figure shows how we can compute the difference for equivalent activities C_1 and C_2.

Definition 17 (different moments). Another special case of a difference between input conditions is the case in which *a* occurs in *P* in no case in which its equivalent occurs in *Q* and (the equivalent of) *a* occurs in *Q* in no case in which *a* occurs in *P*.

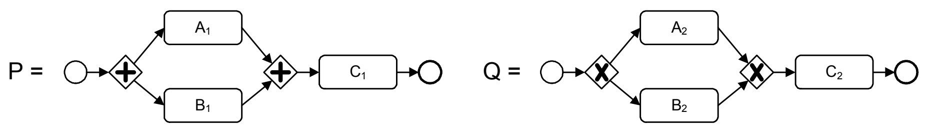
- if *ts* and *us* are the *positive* input conditions for *a* in *P* and *Q*, respectively.
- *a* occurs at different moments in *P* than in *Q* iff
  $ts \cap us = \emptyset$.

Figure 6.iii illustrates the concept of different moments. In this figure we assume that A_1 and A_2, B_1 and B_2, and C_1 and C_2 have equivalent occurrence. The figure shows how we can compute the difference for the equivalent activities C_1 and C_2.

Definition 18 (different start of process). The last special case of difference between input conditions is the case in which *a* can occur from the start in *P*, but not in *Q*, or vice versa.

- if *vs* and *ws* are the input conditions for *a* in *P* and *Q*, respectively.
- if there exists a trace σ_1, such that $[a]_{\sim}\sigma_1 \in vs$, but there exists no trace σ_2, such that $[a]_{\sim}\sigma_2 \in ws$ then *P* has a different start for activity *a*.

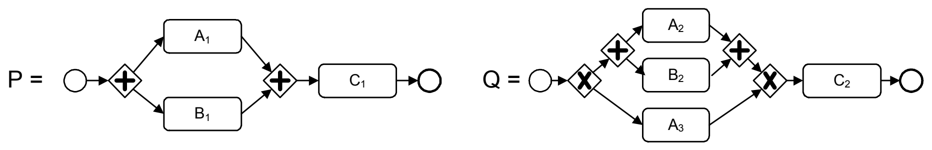
Figure 6.iv illustrates the concept of different start. In this figure we assume that A_1 and A_2 and B_1 and B_2 have equivalent occurrence. The figure shows how we can compute the difference for the equivalent activities B_1 and B_2.



$\text{Tr}(P) = \{A_1\, B_1\, C_1,\ B_1\, A_1\, C_1\}$ $\quad$ $\text{Tr}(Q) = \{A_2\, C_2,\ B_2\, C_2\}$

$vs = \{\{A_1,A_2\}\{B_1,B_2\}\{C_1,C_2\},\ \{B_1,B_2\}\{A_1,A_2\}\{C_1,C_2\}\}$ $\quad$ $ws = \{\{A_1,A_2\}\{C_1,C_2\},\ \{B_1,B_2\}\{C_1,C_2\}\}$

i. Different conditions



$\text{Tr}(P) = \{A_1\, B_1\, C_1,\ B_1\, A_1\, C_1\}$ $\quad$ $\text{Tr}(Q) = \{A_2\, B_2\, C_2,\ B_2\, A_2\, C_2,\ A_3\, C_2\}$

$vs = \{\ \{A_1,A_2,A_3\}\{B_1,B_2\}\{C_1,C_2\},$

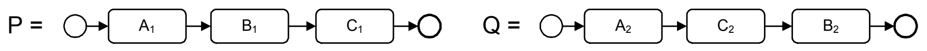
$\{B_1,B_2\}\{A_1,A_2,A_3\}\{C_1,C_2\}\}$

$ws = \{\ \{A_1,A_2,A_3\}\{B_1,B_2\}\{C_1,C_2\},$

$\{B_1,B_2\}\{A_1,A_2,A_3\}\{C_1,C_2\},$

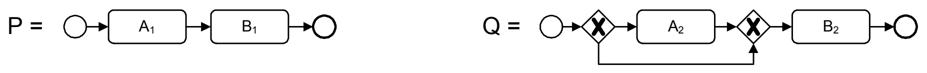
$\{A_1,A_2,A_3\}\{B_1,B_2\}\{C_1,C_2\}\}$

ii. Additional conditions



$[\text{Tr}(P)]_{\sim}/[in_P(C_1)]_{\sim} = \{\{B_1,B_2\}\{C_1,C_2\}\}$ $\quad$ $[\text{Tr}(Q)]_{\sim}/[in_Q(C_2)]_{\sim} = \{\{A_1,A_2\}\{C_1,C_2\}\}$

iii. Different moments



$[\text{Tr}(P)]_{\sim}/[in_P(B_1)]_{\sim} = \{\{A_1,A_2\}\{B_1,B_2\}\}$ $\quad$ $[\text{Tr}(Q)]_{\sim}/[in_Q(B_2)]_{\sim} = \{\{A_1,A_2\}\{B_1,B_2\},\ \{B_1,B_2\}\}$

iv. Different start

Figure 6. Input Conditions

Iterative vs. once-off occurrence. We distinguish a difference between an activity that can occur only once in one process and its equivalent that can occur repeatedly in another process.

Definition 19 (iterative vs. once-off occurrence). If activity a appears in a trace σ that brings process P from a state back into the same state, but not process Q, then a occurs iteratively in process P, but once-off in process Q.

- a occurs iteratively in process P but once-off in process Q iff
  there exists a state s of P and a trace σ in which $[a]_{\sim}$ appears such that $s \Rightarrow^{\sigma} s$,
  while there is no state r of Q and trace ρ in which $[a]_{\sim}$ appears such that $r \Rightarrow^{\rho} r$.

4.2 Detection of Procedural Differences

Since we restrict process models to process models that have a finite set of states, our abstract process model corresponds to a non-deterministic finite state automaton with 'empty string' moves (also known as NFA-ε), where we consider a τ transition to correspond to the 'empty string' move.

An NFA-ε can be used to represent a set of traces, known as its language. We will use the property that an abstract process model with a finite set of states is an NFA-ε and show how the procedural differences can be detected in an NFA-ε. What remains to be shown are the relations between operations on abstract process models and operations on traces, which we used to define the procedural differences.

Lemma 1. If *P* represents the set of traces *Tr*(*P*), then *P*', which can be obtained from *P* by labelling all transitions on action *a* as transitions on action *b*, represents the set of traces *Tr*(*P*'), which can be obtained from *Tr*(*P*) by replacing all occurrences of action *a* with occurrences of action *b*.

Proof. Via the construction of the set of strings accepted by *P*: in a state in which *P* can take transition *a* leading to the acceptance of strings that have *a* at position *p*, *P*' can take transition *b* leading to the acceptance of strings that have *b* at the same position.

Lemma 2. If *P* represents the set of traces *Tr*(*P*), then *P*', which is obtained from *P* by labelling all transitions on action *a* silent, represents the set of traces *Tr*(*P'*), which can be obtained from *Tr*(*P*) by removing all occurrences of action *a*.

Proof. Via the construction of the set of strings accepted by *P*: in any state in which *P* can take transition *a* leading to the acceptance of strings that have *a* and transitioning into a state *s* from which it can continue processing, *P*' cannot take transition *a*, but can continue processing as if it were in state *s*, leading to the acceptance of strings that do not have *a* but are otherwise unchanged.

Lemma 3. The set of dependencies of *a* in *Tr*(*P*) is *vs*, where $x \in vs$ iff there exists *s*, $s' \in S_P$, such that $s \Rightarrow^x s' \rightarrow^a$.

Proof. By definition, a trace $\sigma = \sigma'\ x\ a\ \sigma''$ can only exist if there exists s, $s' \in S_P$, such that $s \Rightarrow^x s' \rightarrow^a$. Vice versa, if there exists s, $s' \in S_P$, such that $s \Rightarrow^x s' \rightarrow^a$ then there also exists a trace $\sigma = \sigma'\ x\ a\ \sigma''$, because of the soundness property: it must be possible for $s \Rightarrow^x s'$ and $s' \rightarrow^a$ to occur (i.e. there must exist a trace σ' such that $s_0 \Rightarrow^{\sigma'} s$) and when they occur this must lead to a final state (i.e. there must exist a state *s*'' and a final state s_f a trace σ'' such that $s' \rightarrow^a s'' \Rightarrow^{\sigma''} s_f$).

Because of lemmas 1, 2 and 3, we can obtain the set of input dependencies of $[a]_\sim$ in $[Tr(P)]_\sim$ / $[in_P(a)]_\sim$, by obtaining the set of input dependencies of $[a]_\sim$ in $[P]_\sim$ / $[in_P(a)]_\sim$ (where $[P]_\sim$ is defined as the replacement of any action *b* in *P* by the action $[b]_\sim$). Since the set of states in *P* is finite, these input dependencies can be found. We can use the definition of input dependencies to compute differences between processes with respect to input dependencies.

The following properties of NFA-ε are well-known and proven [31]:

- A deterministic finite state automaton (DFA) $P = (S, \Sigma, \rightarrow, s_0, S_f)$ is an automaton without ‘empty string’ moves that satisfies the property that for each combination $s \in S$, $a \in \Sigma$, there exists at most one $s' \in S$, such that $s \rightarrow^a s'$.

- For an NFA-ε, P, it is possible to compute a DFA, P', such that $Tr(P') = Tr(P)$.

- For a deterministic finite state automaton (DFA) $P = (S, \Sigma, \rightarrow, s_0, S_f)$, a DFA P^C that represents the language $Tr(P^C) = \Sigma^* - Tr(P)$ is $P^C = (S, \Sigma, \rightarrow, s_0, S - S_f)$.

- For an NFA-ε $P = (S_P, \Sigma_P, \rightarrow_P, s_{0P}, S_{fP})$ and $Q = (S_Q, \Sigma_Q, \rightarrow_Q, s_{0Q}, S_{fQ})$ an automaton $P \cup Q$ that represents the language $Tr(P) \cup Tr(P)$ is $(S, \Sigma, \rightarrow, s_0, S_f)$, such that:
  - $s_0 \notin S_P \cup S_Q$
  - $S = S_P \cup S_Q \cup \{s_0\}$
  - $\Sigma = \Sigma_P \cup \Sigma_Q$
  - $\rightarrow = \rightarrow_P \cup \rightarrow_Q \cup \{s_0 \rightarrow^\tau s_{0P}, s_0 \rightarrow^\tau s_{0Q}\}$
  - $S_f = S_{fP} \cup S_{fQ}$

- For a set of traces that can be represented by a regular expression, it is possible to construct an NFA-ε that represents that set of traces.

We can use these properties to represent the (positive) input condition of some activity a in a process P, in terms of an automaton.

Proof. The input condition of a in P is represented by the NFA-ε: $[P]_\sim$ / $[in_P(a)]_\sim$, which can be constructed using lemma’s 1 and 2. We call this automaton $ic_P(a)$. The positive input condition is defined as the subset of traces of in which a occurs. It corresponds to: $Tr(ic_P(a)) \cap Tr(OccA)$, where $Tr(OccA)$ is the set of traces in which a occurs, which can be represented by the regular expression $(\Sigma_P - \{a\})^* a(\Sigma_P)^*$. Since $Tr(OccA)$ is regular, the NFA-ε, $OccA$, that represents it can be constructed. Now since $Tr(ic_P(a)) \cap Tr(OccA) = (Tr(ic_P(a))^C \cup Tr(OccA)^C)^C$, it follows that the automaton $(ic_P(a)^C \cup OccA^C)^C$, which represents this set of traces, can be constructed.

We can use automata that represent input conditions to determine if two processes have different input conditions for an activity a or if one process has a more strict input condition for an activity a.

If $ic_P(a)$ represents the input condition of a in P and $ic_Q(a)$ represents the input condition of a in Q, we can determine whether $Tr(ic_P(a)) \neq Tr(ic_Q(a))$ as follows. $Tr(ic_P(a)) \neq Tr(ic_Q(a))$ is equal to the statement $(Tr(ic_P(a))^C \cup Tr(ic_Q(a))^C)^C = \emptyset \wedge Tr(ic_P(a)) \neq \emptyset \wedge Tr(ic_Q(a)) \neq \emptyset$, which, using the properties of automata above, is equal to the statement $Tr((ic_P(a)^C \cup ic_Q(a)^C)^C) = \emptyset \wedge Tr(ic_P(a)) \neq \emptyset \wedge Tr(ic_Q(a)) \neq \emptyset$. This statement can easily be checked by constructing the automata and checking whether they represent the empty set of traces (which is the case if, whenever there is a path from the initial to a final state, this path consists only of silent transitions).

We can determine whether $Tr(ic_P(a)) \subset Tr(ic_Q(a))$ as follows. $Tr(ic_P(a)) \subset Tr(ic_Q(a))$ is equal to the statement $Tr((ic_P(a)^C \cup ic_Q(a))^C) = \emptyset \wedge Tr(ic_P(a)) \neq Tr(ic_Q(a))$. $Tr(ic_P(a)) \neq Tr(ic_Q(a))$ can be checked using the procedure from the previous paragraph and $Tr((ic_P(a)^C \cup ic_Q(a))^C) = \emptyset$ can be checked by constructing the automaton an checking whether it represents the empty set of traces.

An iterative vs. once-off difference between processes *P* and *Q* can be detected by noting that *P* and *Q* are connected graphs. Finding the set of cycles (the cycle space) in those graphs is a well-known problem in graph theory. They can be found by finding a spanning tree in the graph. Each edge that is in the graph, but not in the spanning tree forms a fundamental cycle when combined with the tree. Together, the fundamental cycles form a basis for the cycle space.

4.3 Relations between Procedural Differences

There exist implication relations between the procedural differences, such that if one procedural difference exists another procedural difference will always exist ($diff_1 \rightarrow diff_2$). It is good to know where these relations are, such that we can avoid reporting 'logical' differences as much as possible. For example, if $diff_1$ and $diff_1 \rightarrow diff_2$ then we can avoid reporting $diff_2$. Figure 7 summarizes the implication relations between the differences.

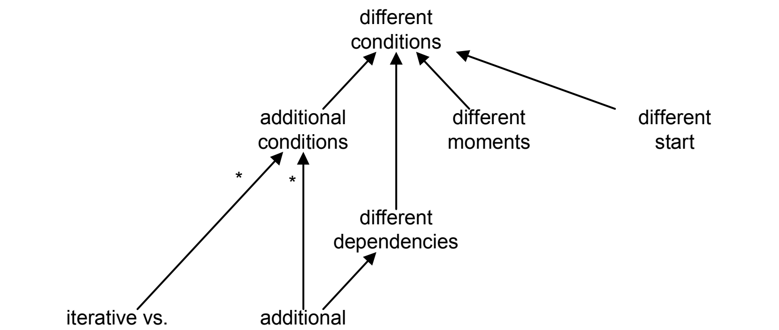

different conditions
additional conditions
different moments
different start
different dependencies
iterative vs. once-off
additional dependencies
*
*
skipped activity
additional dependencies, different dependencies, additional conditions, different moments, different conditions
* if there is an additional conditions (there may not be), it is caused by the other difference

Figure 7. Relations between differences

The relations between additional dependencies and different dependencies, additional conditions and different conditions and between different start and different conditions follow directly from their respective definitions.

If *a* is skipped in process *P* with respect to process *Q*, then there is no trace σ in $[Tr(P)]_\sim$ / $[in_P(a)]_\sim$ such that σ = σ' $[a]_\sim$ σ" (for some σ', σ"). Therefore the set of input dependencies *xs* of *a* in *P* is empty. However, the set of input dependencies *ys* of *a* in *Q* is only empty if the only condition for the occurrence of *a* is the start condition (i.e. only $[a]_{\sim}\ \sigma'$ in $[Tr(Q)]_{\sim}\ /\ [in_Q(a)]_{\sim}$ for some σ'). Hence a skipped activity implies additional dependencies and different dependencies, under the condition that there are other conditions for the occurrence of the skipped activity than the start condition. Using similar reasoning a skipped activity implies different conditions, additional conditions and different moments. Also, if another activity depends on a skipped activity, then this may cause various differences.

Because obviously skipped activities cause a plethora of other differences between processes, we chose not to consider them when checking for other differences. To do this, skipped activities are labelled silent (τ) before checking for other activities.

An activity occurs at a different moment in *P* than in *Q* if its sets of positive conditions in *P* and *Q*, *ts* and *us* respectively, are disjoint: $ts \cap us = \emptyset$. The sets of positive conditions are the intersection between the sets of conditions (*vs* in *P* and *ws* in *Q*) and the set of all possible traces in which the activity occurs: *as*. Hence $ts = vs \cap as$ and $us = ws \cap as$. Therefore, different moments ($ts \cap us = \emptyset$) means that

$(vs \cap as) \cap (ws \cap as) = \emptyset$

$\Rightarrow$ (because of soundness we know that $(vs \cap as) \neq \emptyset$ and $(ws \cap as) \neq \emptyset$)

$(vs \cap as) \neq (ws \cap as)$

$\Rightarrow$

$vs \neq ws$

and, therefore, that there are different conditions as well.

Suppose the sets of conditions of *a* in *P* and *a* in *Q* are *vs* and *ws* respectively. Then we can define the sets of dependencies of *a* in *P* and *a* in *Q* as $xs = \{x \mid \sigma\ x\ [a]_{\sim}\ \sigma' \in vs\}$ and $ys = \{y \mid \rho\ y\ [a]_{\sim}\ \rho' \in ws\}$ respectively. So, clearly, if $xs \neq ys$ then also $vs \neq ws$. Meaning that if there is a different dependency, then there also is a different condition.

Now, if there is an additional dependency for *a* in *Q* then

$xs \subset ys$

$\Rightarrow$

for all $\sigma\ x\ [a]_{\sim}\ \sigma' \in vs$ there exists a $\rho\ x\ [a]_{\sim}\ \rho' \in ws$ and

there exists a $\rho\ x\ [a]_{\sim}\ \rho' \in ws$ such that there is no $\sigma\ x\ [a]_{\sim}\ \sigma' \in vs$

$\Rightarrow$

$vs \neq ws$ and, if the traces in *vs* do not differ from those in *ws* at another position (than the position directly preceding *a*), $vs \subset ws$.

Therefore, there is a different condition as well. If the conditions of *a* in *P* do not differ from those in *Q* at other positions (than the position directly preceding *a*) then there is an additional conditions. Hence, since an additional conditions in *a* will be the result of an additional dependencies, we do not report it. The implication of additional dependencies for *a* in *P* is determined in the same way.

If *a* appears in a trace in *P* and not in *Q*, then there is a trace in *P* in *a* appears infinitely many times while there is no such trace in *Q*. Clearly this leads to different conditions. If there are no other differences, this will lead to additional conditions. Hence we do not report additional conditions in this case either. The implication of iterative versus once-off for *a* in *Q* is determined in the same way.

4.4 Running Example Continued

Figure 8 illustrates our techniques for detecting differences by applying them to the running example. Three differences can be found: one additional cycle, one additional dependency and one skipped activity.

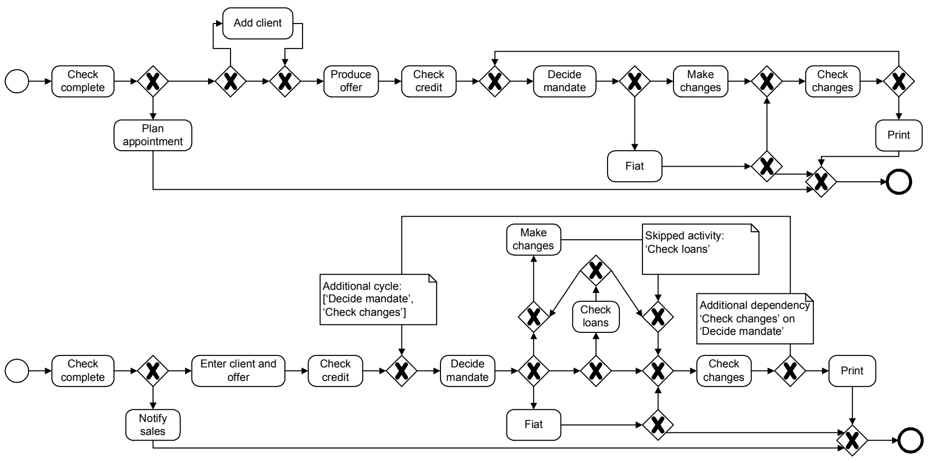


Figure 8. Differences in the Running Example

5 Case Study

To show the practical applicability of our approach we analysed the differences between processes at a financial services provider. We studied 5 processes, each of which involved on average 10 activities and 2 roles. The processes were studied in 2 departments that had the same function, but were originally parts of different organizations. The goal of the analysis was to discover and resolve the differences between the processes in order to merge the departments into a single department.

First, we defined the similarities between the processes. Table 1 summarizes the non-trivial similarities (similarities other than equivalence) that we found in the processes. Second, we analysed the differences between the processes. Table 2 summarizes the result of this exercise. The case study used a slightly larger collection of differences [11]. Here we only report differences that are described in this paper.

Table 1. Similarities (other than equivalence) in case study

Similarity	Instances found
Interchanged activities	7
Refined activity	2
Partly corresponding collections of activities	1

Table 2. Differences in case study

Difference	Instances found
Skipped activity	14
Different dependencies	1
Additional dependencies	2
Activities at different moments in processes	1
Iterative vs. once-off occurrence	3
Different conditions for occurrence	2
Different start of process	3

The case study shows that it is possible in practice to distinguish differences in non-equivalent processes. These differences provide business process analysts with important feedback, since they can help answer the question where they should modify the existing processes to accomplish the merger. The case study also shows that our classification of differences if not exhaustive for practical purposes, because we identified three types of differences that could not be classified according to our classification [11]. However, the differences that *can* be identified remain valuable.

6 Differences and Non-Equivalence

This section proves that the techniques outlined in the previous section only return a difference between two processes if those processes are not trace equivalent. Since section 4.3 shows that all differences imply a 'different conditions' difference, it is sufficient to show that: if there is a 'different conditions' between two processes then the two processes are not equivalent. We can assume that the two processes have the same alphabet, because we replaced each activity by its set of equivalent activities and removed each activity for which there was no equivalent in the other process (and reported it as a skipped activity).

Lemma 4. For any trace $\sigma\in\Sigma^*$, process P, and set of traces $Tr(P)$ of P it holds that:
if $\sigma\in Tr(P)$ then $\forall x\in\Sigma$: $\sigma/\ in_P(x) \in Tr(P)/in(x)$.

Proof. This follows from the definition of trace restriction (definition 11).

Using this lemma we can easily prove that if two processes are trace equivalent ($Tr(P) = Tr(Q)$), then there is no 'different conditions' ($\forall x\in\Sigma$: $Tr(P)/in_P(x) = Tr(Q)/in_Q(x)$).

Proof.

$$Tr(P) = Tr(Q)$$
$$\Leftrightarrow$$
$$\sigma \in Tr(P) \leftrightarrow \sigma \in Tr(Q)$$
$$\Rightarrow \text{(because of lemma 4)}$$
$$\forall x\in\Sigma\text{: } \sigma/\ in_P(x) \in Tr(P)/in(x) \leftrightarrow \forall x\in\Sigma\text{: } \sigma/\ in_Q(x) \in Tr(Q)/in(x)$$
$$\Leftrightarrow$$
$$\forall x\in\Sigma\text{: } \sigma/\ in_P(x) \in Tr(P)/in(x) \leftrightarrow \sigma/\ in_Q(x) \in Tr(Q)/in(x)$$
$$\Leftrightarrow$$
$$\forall x\in\Sigma\text{: } Tr(P)/in(x) = Tr(Q)/in(x)$$

A limitation of the proof is that it only proves that our techniques only return a difference if two processes are not trace equivalent. It does not prove that we will return a difference if the processes are not trace equivalent. We have found one example of a situation in which two processes are not trace equivalent, but our techniques do not return a difference. However, this example involves a process that can perform the empty trace and that situation can easily be solved.

Experimentation has given us the impression that there are not many interesting cases in which processes are not trace equivalent, while our techniques do not return a difference.

7 Related Work

Our work is related to the large body of research in the area of equivalence and difference between business processes.

The work in this paper is based on the formal notions of behavioural equivalence. A survey on different notions of behavioural equivalence is given in [18]. Essentially, two processes have a 'difference' if they are 'non-equivalent' according to one of these notions. Also, work on detecting differences based on the non-equivalence exists [9]. Our work contributes to the work on behavioural equivalence by providing more feedback as to *where* and *why* processes are not equivalent. We do this by classifying non-equivalences. More loose forms of (partial) equivalence, such as behavioural compatibility [7,16,22,23] and behaviour inheritance [3] have a similar relation to our work. Moreover, [2] shows how behaviour inheritance can be used to detect differences between a prescribed process and a process as it is performed in an organization. Differences with respect to compatibility of processes are given in [6,14].

Only recently detection of non-equivalence is being approached from the angle described in this paper, from which differences between processes are classified and detection techniques for classes of differences are defined. Benatallah et al. [6] and Dumas et al. [14] make a classification of differences between interacting business processes. Motari Nezhad et al. [25] also define detection techniques for such differences. Our work differs from theirs in that we focus on similar business processes, while their work focuses on interacting business processes. Each of their 'differences' corresponds to an adapter pattern that will resolve the difference in communication.

A related approach is used in the area of *business processes integration* [17,19,24,27]. However, the work in that area focuses on the merging business processes in spite of their differences. Our work focuses on detecting differences. When it comes to detecting these differences, [27] does identify differences, which they call heterogeneities. [17,19] classify correspondences between business processes. Such a classification also inspires the classification of differences.

The area of *business process evolution* [4,15,8,28,29] deals with evolutionary changes in a business process specification. Such changes result in differences between the original and the evolved business process. (Some of these differences can be recognized in the differences above.) However, the goal of workflow evolution is to develop evolution approaches that keep the differences to a minimum. We accept all possible differences that can arise.

Business process reference models [21] are standard business process models that can be tailored to the needs of a specific company. It is useful to identify *how* reference models can be tailored to the client, which can be done by describing configuration options for the reference models [1,30]. These options pre-define what differences can exist between clients and are therefore related to our classification.

Current tools for business process modelling and analysis also have functionality to analyze differences between processes (we looked at Oracle's Business Process Analysis Suite and MetaStorm's ProVision). This functionality focuses on facilitating multiple people that work on the same process repository. It shows updates of processes (additions, deletions and modifications) since the last synchronization with the process repository.

8 Conclusions and Future Work

This paper presents techniques that help find the procedural differences between business processes that are not (completed trace) equivalent. Procedural differences are differences concerning the behavioural aspect of the processes. We implemented the techniques in a tool [10].

The techniques help find the differences by reporting them in the terms in which the business processes are defined (activities and relations). They also help by providing diagnostic information about each difference by indicating what type of difference it is.

The paper shows that differences are only returned in case two processes are not (completed trace) equivalent. The practical relevance of the techniques is illustrated by showing their application to a case study.

A limitation of the techniques is that they do not detect all possible procedural differences between business processes, because:

- the techniques are defined on the notion of completed trace equivalence, while it is known that stronger notions of equivalence (notions that detect more differences) exist [18];
- the techniques make assumptions about the semantics underlying the business processes, in particular they assume that a semantics exists in terms of a finite state space, while such a semantics may not exist or be possible; and
- we have not proven that a difference will always be returned if two processes are not (completed trace) equivalent. Moreover, we know of a situation in which no difference will be returned for two processes that are not equivalent.

However, even if our techniques cannot detect all possible differences between business processes, the differences that we can detect remain useful. Moreover, based on experience with business processes from practice, we claim that there will not be many practical cases in which no difference will be returned for two non (completed trace) equivalent processes.

We are currently applying our techniques to more case studies with the goal of refining them further. We are also investigating application of the techniques to construct configurable reference models based on differences found between non-configurable reference models and we are investigating application of the techniques to detect incompatibility in the interaction between business processes. Finally, we aim to expand our work to other aspects of business processes. In particular we want to investigate differences between the use of resources in business process and differences between the use of information in business processes.

Acknowledgements

The author thanks Marlon Dumas and Arthur ter Hofstede for the discussions that led to this paper. The author thanks Marlon Dumas and Marian Müllers for reviewing earlier versions of this paper. The author thanks Marian Müllers and Marijn Nagelkerke for testing the tool.

References

1. W. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M. Jansen-Vullers. Configurable process models as a basis for reference modeling. In A. Haller, C. Bussler: *BPM Workshops*, LNCS 3812, Springer-Verlag, Berlin, Germany, 2006, pp. 512-518.
2. W. van der Aalst. Business Alignment: Using Process Mining as a Tool for Delta Analysis and Conformance Testing. *Requirements Engineering* 10, 2005, pp. 198-211.
3. W. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science* 270(1-2), 2002, pp. 125-203.
4. W. van der Aalst, and S. Jablonski. Dealing with Workflow Change: Identification of Issues and Solutions. *International Journal of Computer Systems, Science, and Engineering* 15(5), 2000, pp. 267-276.
5. W. van der Aalst. Verification of Workflow Nets. In: Application and Theory of Petri Nets. Lecture Notes in Computer Science, Vol. 1248. Springer-Verlag, 407-426, 1997.
6. B. Benatallah, F. Casati, D. Grigori, H. R. Motahari Nezhad, and F. Toumani. Developing Adapters for Web Services Integration. In: CAiSE 2005, Springer-Verlag, Berlin, Germany, 2005, pp. 415–429.
7. L. Bordeaux, G. Salaün, D. Berardi, M. Mecella. When are Two Web Services Compatible? In: *Int. Workshop on Technologies for E-Services*. Lecture Notes in Computer Science, Vol. 3324. Springer-Verlag, 2005, pp. 15–28.
8. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data & Knowledge Engineering* 24, 1998, pp. 211-238.
9. R. Cleaveland. On Automatically Explaining Bisimulation Inequivalence. In: *Proceedings of the 2nd Intl. CAV Workshop*, Springer-Verlag , London, UK, 1991, pp. 364-372.
10. R.M. Dijkman. A Tool that Checks Process Alignment. Retrieved October 30, 2007 from: http://is.tm.tue.nl/staff/rdijkman/alignment/tool.html, 2007.
11. R.M. Dijkman. A Classification of Differences in Similar Business Processes. In: Proceedings of the 11th IEEE EDOC Conference (EDOC), IEEE Press, Los Alamitos, CA, USA, 2007, pp. 37-47.

12. R.M. Dijkman, M. Dumas, and C. Ouyang. Formal semantics and automated analysis of BPMN process models (version 2.0). Technical Report Preprint 7115, Queensland University of Technology, Brisbane, Australia, 2007.
13. R.M. Dijkman, and M. Dumas. Service-oriented Design: a Multi-viewpoint Approach. In J. Yang and C. Bussler (guest eds.): International Journal of Cooperative Information Systems (IJCIS), 13(4), pp. 337-368, 2004.
14. M. Dumas, M. Spork, and K. Wang. Adapt or Perish: Algebra and Visual Notation for Interface Adaptation. In: *BPM 2006*, LNCS 4102, Springer-Verlag, Berlin, Germany, 2006, pp. 65-80.
15. C. Ellis, and K. Keddara. A Workflow Change is a Workflow. In: *BPM 2000*, LNCS 1806, Springer-Verlag, Berlin, Germany, 2000, pp. 516–534.
16. H. Foster, S. Uchitel, J. Magee, J. Kramer. Compatibility Verification for Web Service Choreography. In: Proc. of the Int. Conf. on Web Services, 2004.
17. H. Frank, and J. Eder. Towards an Automatic Integration of Statecharts. In: *ER 1999*, LNCS 1728, Berlin, Germany, 1999, pp. 430-444.
18. R. van Glabbeek. The Linear Time – Branching Time Spectrum I: The Semantics of Concrete Sequential Processes. In: *Handbook of Process Algebra*. Elsevier, 2001, pp. 3-99.
19. G. Grossmann, Y. Ren, M. Schrefl, and M. Stumptner. Behavior Based Integration of Composite Business Processes. In: *BPM 2005*, LNCS 3649, Springer-Verlag, Berlin, Germany, 2005, pp. 186-204.
20. B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. *Acta Informatica* 39(3), 2003, pp. 143-209.
21. E. Kindler, and M. Nüttgens (eds.). Workshop on Business Process Reference Models, Nancy, France, September 2005.
22. A. Martens. On Compatibility of Web Services. *Petri Net Newsletter* 65, 2003, pp. 12-20.
23. P. Massuthe, W. Reisig, K. Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics* 1(3), 2005, pp. 35-43.
24. J. Mendling, and C. Simon. Business Process Design by View Integration. In: *BPM 2006 Workshops*, LNCS 4103, Springer-Verlag, Berlin, Germany, 2006, pp. 55-64.
25. H.R. Motahari Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In: *Proceedings of the 16th international WWW conference*, ACM Press, New York, NY, USA, 2007, pp. 993-1002.
26. OMG. Business Process Modeling Notation (BPMN) 1.0, OMG Final Adopted Specification dtc/06-02-01, 2006.
27. G. Preuner, S. Conrad, and M. Schrefl. View Integration of Behavior in Object-Oriented Databases. *Data & Knowledge Engineering* 36(2), 2001, pp. 153-183.
28. M. Reichert, and P. Dadam. ADEPTflex-supporting dynamic changes of workflows without losing control. JIIS 10(2), 1998, pp. 93–129.
29. S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria for Dynamic Changes in Workflow Systems – a Survey. *Data & Knowledge Engineering* 50, 2004, pp. 9-34.
30. M. Rosemann, and W.M.P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems* 32(1), 2007, pp. 1-23.
31. T. Sudkamp. *Languages and Machines (2nd ed.)*. Addison-Wesley, 1996.