Mismatch Patterns in Similar Business Processes

Remco Dijkman

Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
R.M.Dijkman@tue.nl

Abstract. To unify similar business processes, such as processes of similar business units or similar organizations, the similarities and differences between these business processes must be detected and the differences must be resolved. This paper presents a collection of patterns that describe frequently occurring mismatches between similar business processes. These patterns are helpful in the mismatch detection step. We discovered them in practice by comparing processes that we obtained from different business units in two organizations. The patterns help when merging processes in case of a merger between organizations. They also help when merging processes to construct a standardized process that allows organizations that adhere to the standard to interact successfully.

Keywords. Business processes, Modeling, Model Analysis, Process Patterns

1 Introduction

There are many cases in which similar business processes need to be merged partly or fully: when organizations merge it is likely that some of their business processes must be merged as well; when different business units of the same organization start a business unit for processes that they have in common (for example financial processes) these processes must be merged; when a common information system is developed, then the business units involved must at least resolve the differences between the process logic that is implemented by the information system. In each of these cases the similarities and the differences between (parts of) business processes must be identified and the differences must be resolved.

To help identify the differences between business processes, this paper presents a collection of patterns that describe frequently occurring mismatches between business processes. A secondary goal of the paper, which will be pursued further in future work, is to present a complete method to identify and resolve these differences. The focus of this paper is on identifying mismatches between activities in business processes, control-flow relations between these activities and authorization of people to perform these activities.

We obtained our mismatch patterns by comparing the business processes of similar business units. In one comparison we compared 2 business units that had different processes, because they were originally part of different companies. In the other

comparison we compared 3 business units that had different processes, because they had separate operational management that made different decisions regarding the processes. Details about the organizations that we investigated and the mismatches that we discovered are given below.

The remainder of this paper is structured as follows. Section 2 explains how we identify similarities between business processes. It also briefly explains the notation that we use to represent business processes. Section 3 presents the mismatch patterns that we discovered. Section 4 explains the case studies from which we derived our mismatch patterns in detail and summarizes the mismatches that we found in each of these case studies. Section 5 shows possible cases in which the mismatch patterns can be used. Section 5 also shows a complete example in which mismatches between similar business processes are detected based on the patterns from section 3. Section 6 presents related work and section 7 discusses our results and presents our conclusions.

2 Business Processes and Determining their Similarities

This section introduces the concepts and notation that we use to represent business processes. It also explains how and why we identify similarities between business processes before we determine their differences.

2.1 Business processes and notation

In this paper we focus on identifying the differences between activities in business processes, between the control-flow relations that relate these activities and between the authorization of people to perform these activities. We use a simplified form of UML Activity Diagrams [19] to represent these aspects.

Figure 1 shows an example of a business process represented using our notation. We use a filled circle to represent a trigger that starts a process. If there are multiple start triggers then the process can start as a consequence of either one of them. A bulls-eye represents the end of the process. When the process reaches an end there should be no remaining active activities. A rounded rectangle represents an activity in the process. Activities can be related by a control-flow, denoted by an arrow, representing that the target activity (or the end of the process) can occur when the source activity (or the start trigger) has completed. We also use an:

- AND-split, denoted by a vertical bar with one incoming and multiple outgoing control-flows, that represent that all targets can occur when the source has occurred;
- AND-join, denoted by a vertical bar with one outgoing and multiple incoming control-flows, that represent that the target can occur when all sources have occurred;
- XOR-split, denoted by a diamond with one incoming and multiple outgoing control-flows, that represent that represents a choice to allow the occurrence of exactly one target when the source has occurred;

– XOR-join, denoted by a diamond with one outgoing and multiple incoming control-flows, that represent that the target can occur when one of the sources has occurred.

Roles, which represent a group of people with a certain authorization, are represented by rectangles. An activity drawn in the 'lane' of a role denotes that (people in that) role are authorized to perform the activity. Activities can be grouped by drawing a dashed rounded rectangle around them (not part of the UML notation). We use this construct to represent that an activity, which is the collection of activities in the group, is performed by multiple people in different roles.

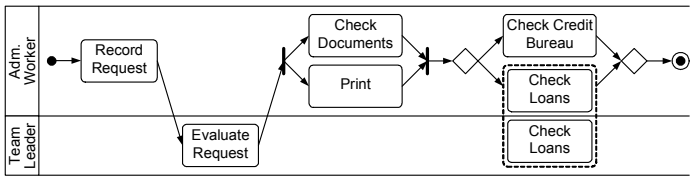

Figure 1. Examples of a business process

2.2 Similarities between business processes

Before we can determine the differences between business processes, we must identify some similarities. This is necessary, because we need some starting point to be able to say more than that the processes are 'totally different'. Therefore, we require that the designer indicates which collections of activities are equivalent and which roles are equivalent for the purpose of the integration.

The requirement that these activities or roles are equivalent *for the purpose of the integration* is important, because depending on the purpose of the integration very different decisions can be made on equivalence. For example, if the purpose of the integration 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 system 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.

To determine equivalence between activities in two processes, we check if the unit of work that they represent is equivalent for the purpose of the integration. We use two criteria to check whether or not two units of work are equivalent:

1. The effect that the units of work will have in the integrated process must be the same (e.g. the same information must be recorded and the same people must be informed of a decision).
2. The way in which the effect is achieved must be the same (e.g. the same information system must be used to record the information and the same means of communication must be used to inform the people).

Depending on the purpose of the integration, either (1) or both (1) and (2) can be used to determine equivalence of units of work.

It is possible that equivalence applies to collections of activities, rather than individual activities. This is the case if two collections of activities represent the same unit of work, while there are no parts of these collections that represent the same unit of work. For example, the activity 'enter client's information' is equivalent to the collection that consists of the activities 'enter name and address information' and 'enter client's partner information', if the unit of work represented by the single activity is the same as the unit of work represented by the collection of activities.

To determine equivalence between roles in two processes, we check if they represent the same authorization in the integrated process (informally: they represent sets of people with the same duties). To perform this check we assume that the organizational structure to which the integrated process applies has already been aligned. Such an alignment requires complex organizational decisions that are outside the scope of this paper. For example, to align the organizational structure, it can be decided that after the integration there will be a single role 'team leader' and that people that were previously authorized to perform the duties of a 'team coach' or a 'unit manager' will be authorized to perform the duties of a 'team leader'. Hence, 'team coaches' and 'unit managers' have the same authorization in the integrated process and, therefore, are equivalent for the purpose of the integration.

The equivalence relation that is established in between activities and between roles in this way must be recorded, for example in the form of a table such as Table 1. In this example, the 'evaluate request' activity from process 'A' is equivalent to the 'evaluate client's application' activity from process 'B' (i.e. the units of work that these activities represent are equivalent with respect to both criteria mentioned above) even though the names of the activities differ. The 'record request' activity from process 'A' has an equivalent effect to the 'scan request' activity from process 'B' (i.e. the units of work that these activities represent are only equivalent with respect to the first criterion mentioned above). The collection of activities from process 'A' mentioned in the last row of the table is equivalent to the collection of activities from process 'B'.

For ease of reading we will use equivalent names to denote equivalent roles or activities in the remainder of this paper. However, in realistic process models equivalent roles or activities most likely do not have the same name, such that the use of an explicit equivalence relation (such as Table 1) is required.

Table 1. Example of an equivalence relation between activities

Activities in process 'A'	Form of equivalence	Activities in process 'B'
'evaluate request'	equivalent to	'evaluate client's application'
'record request'	equivalent effect to	'scan request'
'evaluate loan application', 'evaluate insurance application'	equivalent collection to	'check credit bureau', 'evaluate applications

3 Mismatch Patterns

We differentiate between patterns that represent mismatches concerning the authorization to perform an activity, patterns that represent mismatches concerning

the correspondence between activities and patterns that represent mismatches concerning the flow of control between activities.

3.1 Authorization Mismatch Patterns

A mismatch in the authorization to perform an activity exists if the activity is assigned to different roles in two processes that have to be merged.

Different roles. An authorization mismatch exist if an activity is assigned to one (aligned) role in one process and to another in the other. Figure 2.i shows an example of this type of mismatch.

Single role vs. collection of roles. An authorization mismatch exists if an activity is assigned to a single role in one process and assigned to multiple roles in the other. Figure 2.ii shows an example of this type of mismatch.

Different collections of roles. An authorization mismatch exists if an activity is performed by one collection of roles in one process, to be performed as an interaction between those roles, and to another collection of roles in the other process. Figure 2.iii shows an example of this type of mismatch.

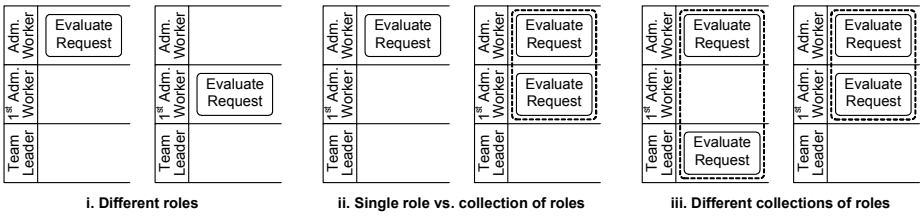

Figure 2. Examples of authorization mismatch patterns

3.2 Activity Mismatch Patterns

A mismatch between activities exists if a unit of work that is represented by a certain collection of activities in one process is represented by a different collection of activities, or not at all, in the other process.

Skipped activity. A skipped activity exists if an activity (that represents a certain unit of work) exists in one process, but no activity representing an equivalent unit of work exists in the other process. Figure 3.i shows an example of this type of mismatch.

Interchanged activities. Interchanged activities exist if an activity exists in one process and an activity that has the same effect (equivalence criteria (1) explained above) but achieves that effect in a different way (equivalence criteria (2) explained above) exists in the other process. This mismatch pattern is only relevant if both the

equivalence criteria explained above determine equivalence, because if only equivalence criteria (1) is used, the activities are equivalent by definition. Figure 3.ii shows an example of this type of mismatch, if recording a request and scanning a request form have the same effect (for example that the client's request is stored in the system).

Refined activity. A refined activity exists if an activity (that represents a certain unit of work) exists in one process, but an equivalent unit of work is only represented by a collection of activities in the other process. We say that the collection of activities *refines* the single activity, because it represents the same unit of work at a different level of granularity. Figure 3.iii shows an example of this type of mismatch, if both the activity and the collection of activities represent the same unit of work.

Corresponding collections of activities. Two processes have corresponding collections of activities, if a collection of activities in one process is equivalent to a collection of activities in the other process (in the sense that they represent an equivalent unit of work), while no subset of activities from one collection is equivalent to a subset of activities from the other collection. Figure 3.iv shows an example of this type of mismatch, if the same unit of work is performed by both collections of activities (for example because the check with the credit bureau is part of the evaluation of the loan application and because the remaining work of evaluating the loan and insurance application corresponds to the work done in the 'evaluate applications' activity). We have not observed this pattern in practice, but we claim that it follows logically from the approach of determining equivalence between activities by comparing the units of work that they represent.

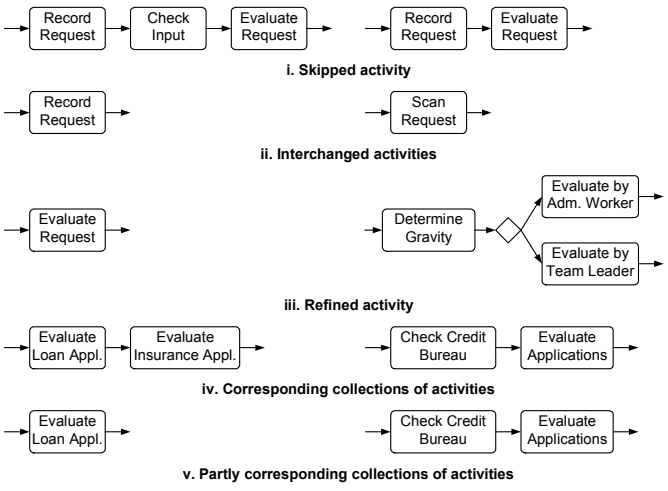

Figure 3. Examples of activity mismatch patterns

Partly corresponding (collections of) activities. Two (collections of) activities are partly equivalent if they partly represent the same unit of work and partly represent different units of work and there is no possibility to re-arrange the activities into

equivalent collections. Figure 3.v shows an example of this type of mismatch, if 'evaluate loan application' includes a check with the credit bureau and 'evaluate applications' includes both the evaluation of a loan application and the evaluation of an insurance application.

3.3 Control-flow Mismatch Patterns

A control-flow mismatch exists if (collections of) activities in one process have different control-flow relations with each other than equivalent (collections of) activities in the other process.

We determine control-flow mismatches, by determining the set of activities on which the occurrence of an activity depends. We do that using the concepts of *closest preceding equivalent activities* and *closest succeeding equivalent activities*. An activity is *preceded* by a *node* if there is a path of control-flow relations from that node to the activity. A node can either be another activity or a control node (AND-split, AND-join, XOR-split, …). The *closest preceding activities* of an activity are all those activities that precede the activity, such that there is no other activity on the path of control-flow relations. An *equivalent* activity is an activity for which there exists an equivalent activity in the other process. The closest preceding *equivalent* activities are all those *equivalent* activities that precede the activity, such that there is no other *equivalent* activity on the path of control-flow relations. The closest succeeding equivalent activities can be determined in a similar manner. An activity depends on all its closest preceding activities (however, for the purpose of comparing processes we should only look at its closest preceding *equivalent* activities). In addition to that, if there is an XOR-split on the path from a closest preceding equivalent activity to the activity itself, then the activity also depends all closest succeeding equivalent activities of that XOR-split. The reason for this is that the activity can no longer occur when one of these activities is performed, because that means that another path was taken from the XOR split and therewith a choice not to perform the activity.

Figure 4 shows an example. If 'check loans', 'reject request' and 'check documents' have equivalents in the other process, then {'check loans', 'check documents'} is the set of closest preceding equivalent activities for 'evaluate request'. 'determine gravity' also precedes 'evaluate request', but it has no equivalent. In addition to {'check loans', 'check documents'}, 'evaluate request' depends on 'reject request', because it is the closest succeeding equivalent activity of the XOR-split on the path from 'check loans' to 'evaluate request'.

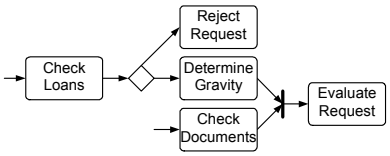

Figure 4. Example of dependency between activities

Different dependencies. We say that equivalent activities from two processes have different dependencies with respect to each other, if the sets of activities on which

they depend differ. Figure 5.i shows an example of this type of mismatch. In the first process ‘evaluate request’ depends on ‘check loans’ and ‘check documents’, while in the second process it depends on ‘check credit bureau’ and ‘check documents.

Additional dependencies. A special case of having different dependencies is the case in which one set of activities includes the other. In this case we say that the set that includes the other has additional dependencies. Figure 5.ii shows an example of this type of mismatch. In the first process ‘make changes’ depends only on ‘evaluate request’, while in the second process it depends on both ‘evaluate request’ and ‘approve evaluation’. Hence, the second set of dependencies includes the first.

Activities occur at different moments in processes. Another special case of having different dependencies is the case in which the sets of activities are disjoint. In this case we say that the activities occur at different moments in their processes. Figure 5.iii shows an example of this type of mismatch. In the first process ‘evaluate request’ depends on ‘record request’, while in the second process it depends on ‘print’. Hence the sets of dependencies are disjoint.

Iterative vs. once-off occurrence. Another special case of having different dependencies is the case in which an activity is part of a loop in one process while it is not in the other process. This means that in one process the activity must be performed correctly in one go, while in the other process it can be performed repeatedly until the result is satisfactory. Figure 5.iv shows an example of this type of mismatch.

Different conditions for occurrence. In case the dependencies for two equivalent activities are the same, the conditions for their occurrence may still differ.

We determine the condition for the occurrence of an activity by drawing a truth table. A truth table specifies when an activity (listed in the last column) can occur as a function of the occurrence of the activities on which it depends (listed in the other columns). A ‘1’ in a column represents that the activity in that column has occurred. A ‘0’ represents that it has not. For each combination of ‘1’s and ‘0’s we determine whether the activity in the last column can occur (represented by a ‘1’), cannot occur (represented by a ‘0’) or if the combination is impossible (represented by a ‘-’).

A condition is impossible, if it is the xor of two or more ‘1’s. This rule applies, because in a well-formed process an XOR-join is always preceded by an XOR-split that causes a choice between two paths, meaning the activities at the end of these paths can never occur both. Considering that an XOR-join represents an xor-condition between the incoming flows and an AND-join represents an and-condition between the incoming flows, we can easily construct a truth table. Table 2 shows an example of a truth table. It is the truth table for the first process from Figure 5.v. It shows that ‘evaluate request’ is allowed to occur if both ‘check documents’ and ‘check loans’ have occurred, but ‘check credit bureau’ has not. It also shows that the situation in which ‘check credit bureau’ and ‘check loans’ have occurred is impossible, because there is an XOR-join on the occurrence of these two activities.

Figure 5.v shows an example of two processes in which the conditions for the occurrence of ‘evaluate request’ differ. For example, in the first process ‘evaluate

request' can occur if 'check loans' and 'check documents' have occurred, while in the second process this situation is impossible. We have not observed this pattern in practice, but we claim that it is a logical superclass of mismatches for the next pattern.

Table 2. A truth table for the occurrence of 'Evaluate Request'

Check Credit Bureau	Check Loans	Check Documents	Evaluate Request
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	-
1	1	1	-

Conditions for occurrence more strict. A special case of different occurrence conditions is the case in which an activity can occur in all the cases that its equivalent in another process can, and more. To test this we do not consider cases that are impossible in either one of the processes, because these cases do not cause the condition to be more or less strict. They merely restrict the cases for which the condition has to be evaluated. We motivate this choice with experience from practice, where people consider an AND-join to be more strict than an XOR-join, even though an XOR-join restricts the set of possible cases. Figure 5.vi shows an example of two processes for which the condition for the occurrence of 'evaluate request' is more strict in the second process, because there it can only occur if both 'check loans' and 'check documents' have been performed, while in the first process it can occur if either one of these have been performed.

Automated Choice. A choice can be made automatically in one process and made manually in another process. A mismatch of this kind cannot be detected automatically. However, an indication of this mismatch is that in one process a skipped activity precedes an XOR-split control node, while in the other process this is not the case. This activity represents making the choice, the result of which causes the choice node to take one path or the other. If the choice is manual, this activity is present; if the choice is automated, it is not, because then the activity of making the choice is embedded in the XOR-split control node itself. Figure 5.vii shows an example in which a choice between two options is performed manually in the first case and automated in the second case.

Different start of process. In case one process allows an activity to be performed from the start of the process, while the other process does not, we say that there is a mismatch in the start of the process. Figure 5.viii shows an example in which the activity 'receive revision' can only occur during the process in the first process, while it can occur from the start of the process in the second process. This mismatch may be caused by one of the processes having more triggers than the other for a particular case. For example, one for the initial requests and one for revised requests.

This mismatch can occur in combination with the mismatch of skipped activities (e.g. process P allows activity c to occur from the start of the process, while the process q requires that a and b occur before c can occur. However, activities a and b are not mentioned in process P. They are skipped activities.). In this case it is

advisable to check if the skipped activities appear in another process, because it can be the case that a designer chose to split-up the processes at a different point. Figure 5.ix shows an example in which processes are split-up differently. In the first process 'payout client' and 'payout intermediary' are part of the acceptance process, while in the second process they are part of a new process.

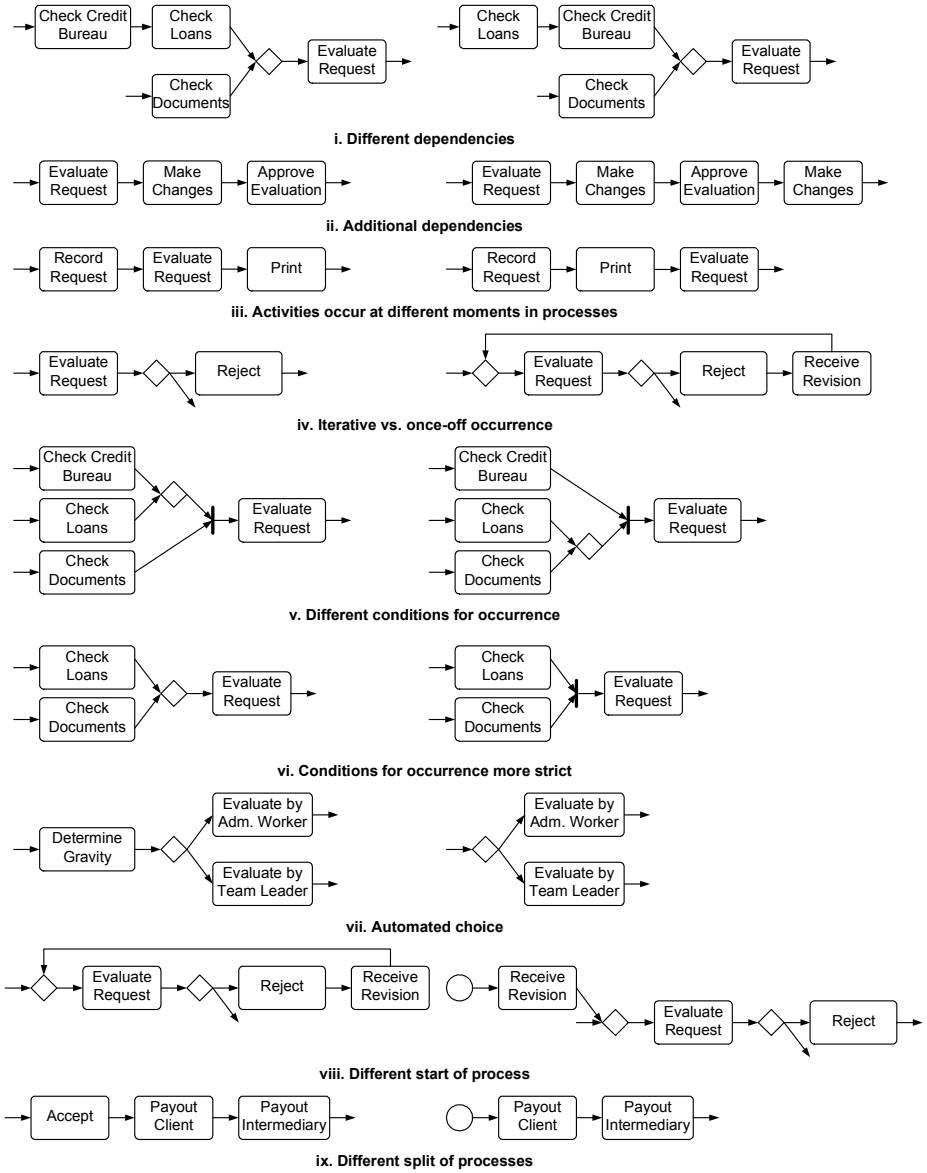

Figure 5. Examples of control-flow mismatch patterns

4 Case Studies

We discovered the mismatch patterns in two case studies.

In the first case study we studied the planning processes at TNT Post, the largest Dutch postal service provider. We studied 10 processes, each of which involved on average 9 activities and 3 roles. The processes were studied in 3 (geographically) different departments that had the same function for a different geographical area in the Netherlands. The most strict notion of equivalence between activities (both effect and means of the activity must be the same) was used. Table 3 summarizes how many instances of each mismatch pattern were found during this case study. Of patterns that are not shown, no instances were found. We counted these instances by using a 'standardized' process and comparing the processes from the departments to this process.

Table 3. Instances of mismatch patterns found in case study 1

Mismatch pattern	Number of Instances found
Different roles	2
Single role vs. collection of roles	3
Different collections of roles	1
Skipped activity	11
Interchanged activities	2

In the second case study we studied some processes at a large internationally operating 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 most strict notion of equivalence between activities (both effect and means of the activity must be the same) was used. Table 4 summarizes how many instances of each mismatch pattern were found during this case study.

Table 4. Instances of mismatch patterns found in case study 2

Mismatch pattern	Number of Instances found
Different roles	3
Skipped activity	13
Interchanged activities	7
Refined activity	2
Partly corresponding collections of activities	1
Different dependencies	1
Additional dependencies	2
Activities occur at different moments in processes	1
Iterative vs. once-off occurrence	3
Conditions for occurrence more strict	2
Automated choice	1
Different start of process	2
Different split of process	1

In addition to the patterns that we found in the case studies and that we could classify, we found three mismatch patterns that we could not classify at this time.

Firstly, our technique for process modeling does not support representing the '4-eyes' principle, according to which information must be generated by a person and then checked by *another* person. Therefore, mismatches with respect to this principle cannot be detected. Arguably, mismatches with respect to this principle are usually detected. Because if a 'check' activity does not exist in one process, while it does

exist in another process, this appears as a 'skipped activity' and in case a 'check' activity exists in both processes it is performed by different people. However, we found one case in which a 'check' activity had to be performed by two people, both different from the person that generated the information. This mismatch could not be detected.

Secondly, our technique for process modeling does not support representing that an activity can be performed by people in one role *or* another role. Therefore, we cannot detect mismatches with respect to this modeling construct. We encountered three instances of this mismatch.

Thirdly, in case an activity in one process is refined by an activity in another process, the behavior of the refined activity as it is determined by its control-flows may differ from the behavior of the other activity. However, because, to compare it, the refined activity it is treated as a single activity with incoming and outgoing flows, such differences are not detected. Figure 6 illustrates this case. The refined activity (comprising of 'check loans' and 'check documents') is treated as a single activity to compare it with the activity 'check'. However, checking loans may be skipped in the refined activity, while this may not be an option in the 'check' activity. We encountered one instance of this mismatch pattern.

Figure 6. Example of a control-flow mismatch in a refined activity

5 Possible Uses and Example

The most obvious use for the mismatch patterns is to use them when merging processes when different organizations or organizational units merge. The patterns can help to precisely identify the differences between the processes and ultimately resolve these differences.

As an example Figure 7 shows the loan application processes of two different organizations. Although these processes concern the same work, there clearly are differences. Both processes start with checking the completeness of a loan application. If the application is complete the process continues, otherwise action is taken. Subsequently the client's information is looked up and entered into the system and an offer is produced. Next the client's creditworthiness with respect to the offer is checked and in the first process some additional checks are performed. After all the checks are performed a decision is made to either allow the administrative worker to make changes to the offer, or to let the offer be approved (and possibly changed) by fiat. In the first process this choice is made automatically. In the second process it is made manually in an activity 'decide mandate'. In that process, there is also the possibility to proceed without making changes or letting the offer be approved by fiat. However, in this case it may be necessary to perform an additional check on the client's existing loans, potentially leading to changes in the offer. After all the checks

are performed and changes are made, the changes are checked, possibly leading to a renewed integral check of the offer. Finally the offer is printed.

The processes differ with respect to the means they use to inform a client that his application is incomplete. In one process the administration plans an appointment with the client, while in the other process administration informs sales of the incomplete application. Hence, these activities are *interchangeable*. Three *skipped* activities exist: 'checking module', 'decide mandate' and 'check loans'. 'Decide mandate' not only is a skipped activity, but also part of a choice that is an *automated choice* in the other process. The 'check changes' activity is performed by *different roles* in both processes. The 'enter client and offer details' in the second process is *refined* by 'check existing', 'add client information' and 'produce offer' in the other process. Also, the 'enter client and offer details' activity has an *additional dependency* on 'fiat' with respect to the activities that refine it. This additional dependency is caused by the fact that in the second process the activity is performed *iteratively, versus once-off* in the first process. Finally, 'check changes' has an *additional dependency* on (the closest preceding equivalent activity) 'check credit'. Interestingly, this is the only control-flow mismatch in the seemingly complex differences between the flows through 'make changes', 'check loans' and 'fiat' activities. This clearly illustrates the use of the patterns as an aid for identifying differences between processes.

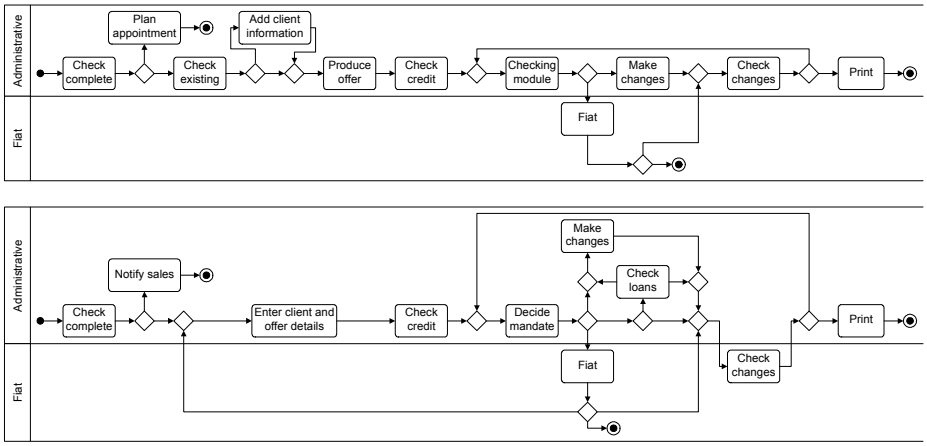

Figure 7. Example of mismatches between processes of two organizations

Other uses for the patterns include: using them to help identify (and resolve) the differences between the views of different stakeholders in the same process, when a model for that process is being constructed; and using them to help resolve differences between processes for which a common enterprise information system must be developed.

A less obvious use for the patterns is using them in standardization activities, in which a standard process must be developed. In that case the mismatch patterns can help identify and resolve the differences between different proposed standard processes. They can also help to identify and resolve the differences between the standard process and the process of a party that wants to conform to the standard.

We consider the reference models proposed by the major ERP vendors (such as the SAP reference model [5]) as de-facto standards. Hence, the development of these reference models can benefit from the patterns. Also, the patterns can help to identify the differences between the reference processes and the processes of a client that wants to implement an ERP system. The differences then provide an indication of the configuration that is needed for the implementation of the system.

6 Related Work

Other areas of research that address differences between business processes are:
- business process integration;
- equivalence of business process behavior;
- business processes evolution; and
- business process reference models.

Our work is closest to the work on *business processes integration* [12,14,18,20] However, the work in that area focuses on the merging business processes in spite of their differences. Our work focuses more on detecting the differences between the processes. When it comes to detecting these differences, [20] does identify a few mismatch patterns, which they call heterogeneities, between business processes. [12,14] classify correspondences between business processes. Such a classification also inspires the classification of mismatches.

The work on business process integration is based on the work on database view integration [9,24], but database view integration focuses on merging information aspects rather than behavioral aspects. Database view integration does address mismatch patterns between (information) models in detail, as well as techniques to solve these mismatches.

Another related area is that of determining *equivalence of business process behavior*. A survey on different forms of behavioral equivalence if given in [13]. The work on behavioral equivalence differs from ours, because our work focuses on determining differences rather than equivalence. However, like with the work on classifying correspondences, the classification of different forms of (non-)equivalence inspires the classification of mismatches. More loose forms of (partial) equivalence, such as behavioral compatibility [16,17] and behavior inheritance [2] have a similar relation to our work. Moreover, [1] shows how behavior inheritance can be used to detect mismatches between a prescribed process and a process as it is performed in an organization. Mismatch patterns with respect to compatibility of processes are given in [4,11].

The area of *business process evolution* [3,8,10,21,22] deals with evolutionary changes in a business process specification. Such changes also result in mismatches (between the original and the evolved business process). In fact some of these mismatches can be recognized in the mismatch patterns above. However, the goal of workflow evolution is to develop evolution approaches that keep the mismatches to a minimum. We accept all possible mismatches that can arise.

Business process reference models [15] 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 [23]. These options pre-define what differences can exist between clients and are therefore related to our mismatch patterns.

7 Conclusion, Discussion and Future Work

This paper presents patterns of mismatches between business processes of (departments of) organizations that have the same function. The patterns were developed by investigating organizations in practice. We have shown how frequently the patterns occurred in the organizations we investigated. We have also shown how the patterns can be used, for example, to resolve mismatches for the purpose of merging organizations.

We do not claim to have discovered all possible mismatch patterns, because:

- we only investigated mismatches between departments in the same organization, while mismatches between different organizations can be more complex;
- we only investigated patterns that appear in processes modeled with a simplified modeling language (one that only allows for AND/XOR-splits and and/or joins), limiting the set of patterns that can be expressed and therefore detected;
- we only studied the mismatches in two organizations;
- we focused on authorization, activity and control-flow aspects of business processes, while other aspects, such as the information aspect, can be considered as well.

However, we do claim that, although the set of patterns is not complete, it is an interestingly large set. Moreover, a set that is helpful when detecting and resolving mismatches, regardless of other mismatches that may exist. These claims are supported by our findings in the case study, because (focusing on the aspects mentioned above), we only detected three mismatch patterns in practice that could not be classified by the patterns from section 3. Also, although our case studies focus on mismatches between processes within an organization, one of the organizations was a merger of different organizations. Therefore, we claim that the patterns that we found include a reasonably large subset of the patterns that can be found between different organizations.

We made two important observations in the case study. Firstly, the ‘different dependencies’ and ‘additional dependencies’ mismatch patterns represent a set of diverse and complex mismatches. This can pose difficulties when trying to propose uniform solutions for mismatches in each of these categories. We could investigate mismatches in these categories to see if further classification of mismatches (and corresponding solutions) is possible. Secondly, we discovered that some mismatches imply that other mismatches exist as well. This means that, when we develop a tool that detects mismatches, several mismatches concerning the same set of activities or roles would be identified. This could confuse the designer using the tool and is also an issue for further study.

One of our aims for future work is to investigate processes from more organizations and to detect mismatches between those processes (possibly leading to

more mismatch patterns). To support this activity and to improve the usability of the patterns, we aim to develop techniques and tools to detect mismatches according to the patterns above. Finally, we aim to develop to resolve mismatches between business processes with the aim of developing standardized processes. Standardized processes can be used as reference models in tools. They can also be used as a basis for developing a collaboration between organizations, in which case we also refer to the standardized processes as choreographies [6,7].

Acknowledgements

The author thanks Marthe Uitterhoeve for collecting the differences between the processes at TNT Post. Also, the author thanks the organizations, as well as the people from those organizations (in particular Frank Knot), that were kind enough to provide information about their processes for this paper.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. T. Curran, and G. Keller. SAP R/3 Business Blueprint. Prentice Hall, Upper Saddle River, NJ, USA, 1998.
6. R.M. Dijkman. Choreography-Based Design of Business Collaborations. BETA Working Paper WP-181, Eindhoven University of Technology, Eindhoven, The Netherlands, 2006.
7. 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), Special Issue on Service Oriented Modeling 13(4), pp. 337-368, 2004.
8. C. Ellis, and K. Keddara. A Workflow Change is a Workflow. In: BPM 2000, LNCS 1806, Springer-Verlag, Berlin, Germany, 2000, pp. 516–534.
9. C. Batini, M. Lenzerini, and S.B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys* 18(4): 323-364, December 1986.
10. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data & Knowledge Engineering* 24, 1998, pp. 211-238.
11. 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.
12. H. Frank, and J. Eder. Towards an Automatic Integration of Statecharts. In: *ER 1999*, LNCS 1728, Berlin, Germany, 1999, pp. 430-444.

13. 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.
14. 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.
15. E. Kindler, and M. Nüttgens (eds.). Workshop on Business Process Reference Models, Nancy, France, September 2005.
16. A. Martens. On Compatibility of Web Services. *Petri Net Newsletter* 65, 2003, pp. 12-20.
17. P. Massuthe, W. Reisig, K. Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics* 1(3), 2005, pp. 35-43.
18. 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.
19. Object Management Group. UML 2.0 Superstructure Specification. Specification ptc/04-10-02, 2004.
20. G. Preuner, S. Conrad, and M. Schrefl. View Integration of Behavior in Object-Oriented Databases. *Data & Knowledge Engineering* 36(2), 2001, pp. 153-183.
21. M. Reichert, and P. Dadam. ADEPTflex-supporting dynamic changes of workflows without losing control. JIIS 10(2), 1998, pp. 93–129.
22. 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.
23. M. Rosemann, and W.M.P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems* 32(1), 2007, pp. 1-23.
24. S. Spaccapietra, C. Parent. View Integration: A Step Forward in Solving Structural Conflicts. *IEEE Transactions on Knowledge and Data Engineering* 6(2): 258-274, April 1994.