BPI Challenge 2012: The Transition System Case

H.M.W. Verbeek

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
h.m.w.verbeek@tue.nl

Abstract. The Transition System Miner, together with the Simple Log Filter and the Transition System Analyzer, is used to investigate the log used for the BPI Challenge 2012. Conclusions are drawn for the control-flow perspective, the date perspective, and the resource perspective, which shows the flexibility of the Miner. The results show that the process as captured in the event log is nicely structured, that it contains hardly any noise, and that the different events (Application, Offer, and Work Item) can be nicely captured by transition systems. Furthermore, it shows that the company who owns the process does not use case managers for handling the applications, as a lot of handover-of-works occur.

1 Introduction

In this paper, we take up the BPI Challenge 2012 [1] using mainly transition systems. We take the original event log, which we simply call $\mathcal{L}$, create appropriate sublogs, use the *Transition System Miner* to mine transition systems from these sublogs and to create subsublogs if appropriate, and use the *Transition System Analyzer* to enrich the resulting transition system with timing information. In the end, we will draw several conclusions on both the control-flow perspective, the data perspective, and the resource perspective.

The remainder of this paper is organized as follows. First, Section 2 introduces the log to be used for this challenge, and where it can be downloaded from. Also, this Section introduces the three techniques that will be used in this paper: The Simple Log Filter, the Transition System Miner, and the Transition System Analyzer. These techniques have all been implemented as plug-ins in ProM 6 [2]. Section 3 investigates the control-flow perspective using only the Application events. Section 4 will do this for the Offer events, and will link some Offer events to some Application events, while Section 5 will do this for Work Item events. Section 6 adds the data perspective into the equation by extending Application control-flows with data attributes as present in the log. Then, Section 7 adds the resource perspective by extending Work Item control flows with data corresponding to the 5 most-frequent resources. Finally, Section 8 concludes the paper.

Event Type	Meaning
States starting with A_	States of the application
States starting with O_	States of the offer belonging to the application
States starting with W_	States of the work item belonging to the application
COMPLETE	The task (of type A_ or O_) is completed
SCHEDULE	The work item (of type W_) is created in the queue (automatic step following manual actions)
START	The work item (of type W_) is obtained by the resource
COMPLETE	The work item (of type W_) is released by the resource and put back in the queue or transferred to another queue (SCHEDULE)

Table 1. Event type explanation

2 Preliminaries

2.1 Log Description

As mentioned on the website[1], the event log $\mathcal{L}$ contains events related to an application process. Table 1 shows an overview of event types and their explanations. An application is submitted through a webpage. Then, some automatic checks are performed, after which the application is complemented with additional information. This information is obtained trough contacting the customer by phone. If an applicant is eligible, an offer is sent to the client by mail. After this offer is received back, it is assessed. When it is incomplete, missing information is added by again contacting the customer. Then a final assessment is done, after which the application is approved and activated.

2.2 Simple Log Filter

The *Simple Log Filter* allows you to filter an event log in 4 ways.

1. It allows you to select an appropriate filtering action for any lifecycle transition present in the log:
   Keep : If an event has a *keep* lifecycle transition, then it will be filtered in.
   Remove : If an event has a *remove* lifecycle transition, then it will be filtered out.
   Discard instance : If a trace contains an event with a *discard instance* lifecycle transition, then it will be filtered out.
2. It allows you to select concept names to filter in for start events. If the first event of a trace has a selected concept name, then it will be filtered in, otherwise it will be filtered out.

[1] See `http://www.win.tue.nl/bpi2012/doku.php?id=challenge`

3. It allows you to select concept names to filter in for end events. If the last event of a trace has a selected concept name, then it will be filtered in, otherwise it will be filtered out.
4. It allows you to select concept names for events. If an event has a selected concept name, then it will be filtered in, otherwise it will be filtered out.

As an example, we have created 3 filtered log from the original log $\mathcal{L}$, where the first filtered log ($\mathcal{L}_A$) contains only application events (concept name starts with A_), the second filtered log ($\mathcal{L}_O$) contains only offer events (O_), and the third filtered log ($\mathcal{L}_W$) contains only work item events (W_). To filter the first log, we selected *keep* for all lifecycle transitions, selected all possible concept names for start events, selected all possible concept names for end events, and selected only the concept names that start with A_. As a result, no traces are filtered out, and only the events that start with A_ are filtered in.

To select the Simple Log Filter in ProM 6, select the action labeled *Filter Log using Simple Heuristics*. This wil start a 4-step wizard that will guide you through the options as mentioned. First, it shows you which lifecycle transitions appear in the log, and it allows you to select keep, remove, or discard instance for every transition. Next, it shows the possible concept names that start traces, and it allows you to select which ones to filter in. Only those cases that start with a selected concept name will be filtered in, the others will be filtered out. For ease of use, the wizard also allows you to select the top $X\%$ of the start concept names, in which case it will select the most frequent names until at least $X\%$ of the traces will be filtered in. Third, in a similar fashion, the wizard allows you to select end concept names. Last, the wizard allows you to select which concept names to filter in. Only selected concept names will be filtered in, other concept names will be filtered out. Note that this step filters out an entire trace if all concept names from that trace are filtered out. Again, you can use a slider to select the top $X\%$ of concept names.

2.3 Transition System Miner

The *Transition System Miner* allows you to mine an event log for a transition system. In the resulting transition system, the transition will correspond to the events in the log, whereas a state will correspond to a situation in between two events.

An important question to be answered by you is what exactly comprises a state, or more specifically, when do two states differ? Basically, the Transition System Miner allows you to specify the identity of a state in a number of ways. First, it allows you to specify that a state is identified by looking at the events preceding the state (the *past events*), by looking at the events succeeding the state (the *future events*), or by a combination of both. For both the past events as the future events, you can select the log classifier that will be used to classify these states. Note that the classifier for the past events may be different from the classifier for the future events. Next, you can select an abstraction which is

to be applied on the resulting event classes. This abstraction consists of a collection type and an optional limit on the size of these collections, where possible collection types are *list*, *bag*, and *set*. The list collection type maintains the order between subsequent event classes and their cardinalities, the bag type maintains only the cardinalities, whereas the set type only maintains the support of event classes. The optional limit limits the size of the collection: If the collection size exceeds the limit, then the event classes farthest away from the state will be removed.

As an additional option, the Transition System Miner allows you to incorporate *unclassified* data attributes into the state identifiers. However, for unclassified data attributes only the last value up to the state (say, the current value of this data attribute) is taken into account. If you need to differentiate states based on multiple values of a data attribute in the past, or based on future values, then you should make this data attribute a classified data attribute (by adding an appropriate classifier to the event log) and you should select this classifier and use it as mentioned above.

Apart from this important configuration of the states, the Transition System Miner allows you to select which concept names to show on the transitions, and which of the following 4 post-mining operations to perform on the resulting transition systems:

Remove self loops : Selecting this option removes any self loop transition (a transition that starts and end in the same state) from the transition system.

Improve diamond structure : Selecting this option completes any unfinished diamond in the transition system. Assume that we have four states (say *North*, *East*, *South* and *West*) and three transitions in the transition system: A from *North* to *East*, A from *West* to *South*, and B from *North* to *West*. Apparently, executing B in the *North* state did not disable the enabled A, which suggests that A and B are in parallel. However, the transition B from *East* to *South* is missing, and existing tools might fail to recognize the suggested parallelism due to the fact that this transition is missing. Selecting this option effectively adds the missing transition B from *East* to *South*.

Merge states with identical inflow Selecting this option merges two states in the transition system that have the same inflow, that is, the same multi-set of incoming transitions. Note that selecting this option ensures that there is only one source state (a state with empty inflow) in the transition system, as all source states will be merged.

Merge states with identical outflow] This is similar to the previous option, but then with the outflow instead of the inflow.

For more details on the Transition System Miner, we refer to [3].

To select the Transition System Miner in ProM 6, select the action labeled *Mine Transition System*. This will start a wizard that will guide you through the options as mentioned. First, it shows you which classifiers are present in the event log, and it allows you to select classifiers for past past events as future events. If you select multiple classifiers for past events or for future events, these classifiers will be combined into a single classifier. Also, it allows you to select

the option to use unclassified data attributes to identify states. Second, it allows you to select the collection type and the optional limit. Note that this collection type and limit will be used for both the past events and the future events. Third, it allows you to select which unclassified data attributes to use to identify states. This step is skipped if the corresponding option in the first step. Fourth, it allows you to select which labels (which are combinations of concept names and lifecycle transitions) should appear on the transitions. Fifth, it allows you to select which post-mining operations should be applied. These operations will be applied as long as the resulting transition system is affected by them. So, in the end, no selected post-mining operation will have an effect anymore. Last, it allows you to review your configuration and to either change it or approve it.

A feature of the Transition System Miner that we will use in this paper is the fact that it can be used to filter an event log. If you have mined an event log for a transition system using the Transition System Miner, then you can select any number of states and/or transitions and filter the corresponding event log on these states and/or transitions. You only need to provide a threshold of how many of the selected states and/or transitions should be matched by any trace. Any trace that actually matches this threshold will be filtered in, any trace that does not match this threshold will be filtered out. By selecting a threshold of 1, you can filter in any trace that matches at least one of the selected states and/or transitions. By selecting a maximal threshold, you can filter in any trace that matches all selected states and/or transitions. By selecting any other threshold you can filter in any trace that matches some selected states and/or transitions.

2.4 Transition System Analyzer

The *Transition System Analyzer* allows you to enrich a transition system as mined by the Transition System Miner with aggregated time information from the event log. In the resulting transition system, states will have frequencies and aggregated sojourn times, elapsed times, and remaining times, and transitions will have frequencies and aggregated durations.

The states and transitions will be colored to signal possible bottlenecks in the transitions system. First, the Analyzer determines the maximal and minimal time any case spends in some state, say s_{max} and s_{min}, and the maximal and minimal time any transition takes (say, t_{max} and t_{min}). Second, it colors the states and transitions accordingly. For example, If the time spend in a state exceeds $s_{min} + 0.8 * (s_{max} - s_{min})$, then it is colored red, otherwise if it exceeds $s_{min} + 0.6 * (s_{max} - s_{min})$, then it will be colored yellow, otherwise it will be colored blue. Using a slider, you can change the thresholds (0.8 and 0.6).

If you select a state or a transition, its frequency and aggregated times are shown on the screen.

To select the Transition System Analyzer in ProM 6, select the action labeled *Analyze Transition System.*

State	Sojourn time	Frequency
A_ACCEPTED	47min 34s	5113
A_ACTIVATED	0ms	476
A_ACTIVATED+A_APPROVED	0 ms	787
A_ACTIVATED+A_APPROVED+A_REGISTERED	0ms	2246
A_ACTIVATED+A_REGISTERED	0ms	1122
A_APPROVED	1ms	1055
A_APPROVED+A_REGISTERED	0ms	337
A_CANCELLED 1	0ms	1
A_CANCELLED 2	0ms	1100
A_CANCELLED 3	0ms	66
A_CANCELLED 4	0ms	1640
A_DECLINED 1	0ms	5719
A_DECLINED 2	0ms	1085
A_DECLINED 3	0ms	29
A_DECLINED 4	0ms	802
A_FINALIZED	16d 16h	5015
A_PARTLYSUBMITTED	2h 25min	13087
A_PREACCEPTED	2d 19h	7367
A_REGISTERED	0ms	715
A_SUBMITTED	581ms	13087

Table 2. Sojourn times for the Application states

3 Application states

A quick inspection of the log $\mathcal{L}_A$ shows that there are 10 event classes, while the longest trace contains only 8 events. This leads us to believe that this log does not contain any recurrent behavior. In such cases, the bag abstraction often works best, as it allows you to detect parallel behavior. Therefore, we have executed the Transition System Miner on the log $\mathcal{L}_A$ using the following settings:

- Use *Activity* classifier for past events, no classifier for future events.
- Use unlimited bag abstraction.
- Use all event classes.
- Use no transition labels.
- Use no post-mining operations.

Figure 1 shows the resulting transition system, after having extended the transition system using the Transition System Analyzer. For sake of completeness we mention that we have manually added comprehensive state labels. A dotted line (see the topmost state) indicates that a state is the start state, whereas an extra line (see for example the bottommost state) indicates that a state is an accept (or final) state. Table 2 and Table 3 show the corresponding timing information, as obtained by the Transition System Analyzer.

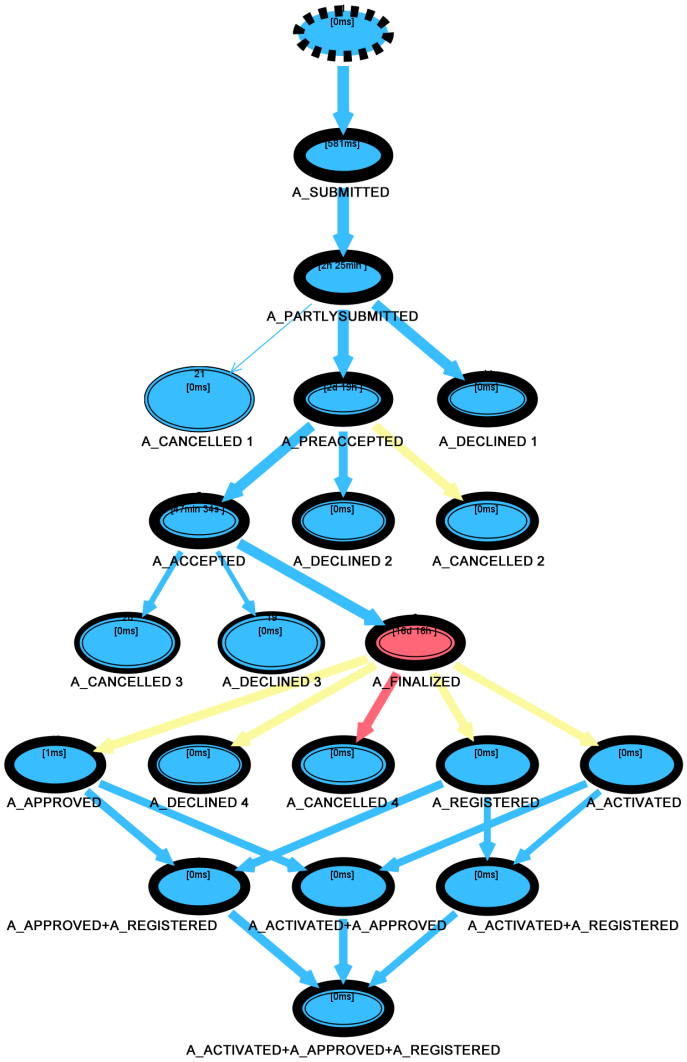

Fig. 1. Annotated transition system for Application states using unlimited bag abstraction.

From the transition system and the timing information, we can conclude the following:

- The process starts with states A_SUBMITTED and A_PARTLYSUBMITTED.
- A successful application ends with A_ACTIVATED, A_APPROVED, and A_REGISTERED, which occur in parallel, preceded by A_FINALIZED.
- From the 13087 applications, 2246 ended successfully, 2807 were canceled, 7635 were declined, and 399 were abandoned (69 in the A_PREACCEPTED state, 3 in the A_ACCEPTED state, 327 in the A_FINALIZED state).

From State	To State	Duration	Frequency
	A_SUBMITTED	0ms	13087
A_ACCEPTED	A_CANCELLED 3	2d	66
A_ACCEPTED	A_DECLINED 3	3h 13min	29
A_ACCEPTED	A_FINALIZED	9min 25s	5015
A_ACTIVATED	A_ACTIVIATED +A_APPROVED	0ms	322
A_ACTIVATED	A_ACTIVIATED +A_REGISTERED	0ms	154
A_ACTIVATED +A_APPROVED	A_ACTIVATED +A_APPROVED +A_REGISTERED	0ms	787
A_ACTIVATED +A_REGISTERED	A_ACTIVATED +A_APPROVED +A_REGISTERED	0ms	337
A_APPROVED	A_ACTIVATED +A_APPROVED	2ms	465
A_APPROVED	A_APPROVED +A_REGISTERED	0ms	590
A_APPROVED +A_REGISTERED	A_ACTIVATED +A_APPROVED +A_REGISTERED	0ms	1122
A_FINALIZED	A_ACTIVATED	15d 17h	476
A_FINALIZED	A_APPROVED	16d 4h	1055
A_FINALIZED	A_CANCELED 4	21d 12h	1640
A_FINALIZED	A_DECLINED	15d 11h	802
A_FINALIZED	A_REGISTERED	15d 22h	715
A_PARTLYSUBMITTED	A_CANCELLED 1	12h 14min	1
A_PARTLYSUBMITTED	A_DECLINED 1	2h 52min	5719
A_PARTLYSUBMITTED	A_PREACCEPTED	2h 4min	7367
A_PREACCEPTED	A_CANCELLED 2	13d 16h	1100
A_PREACCEPTED	A_DECLINED 2	1d 10h	1085
A_PREACCEPTED	A_ACCEPTED	18h 41min	5113
A_REGISTERED	A_ACTIVATED +A_REGISTERED	0ms	183
A_REGISTERED	A_APPROVED +A_REGISTERED	0ms	532
A_SUBMITTED	A_PARTLYSUBMITTED	581ms	13087

Table 3. Sojourn times for the Application transitions

– The total log contains events from 25 weeks, where the last 3 weeks are almost empty. The first abandonnement takes place early week 14[2], so it might be that these abandonnements correspond to unfinished cases.

[2] This can be checked easily by animating the mined transition system, and checking when the first red tag appears on these supposedly intermediate states.

State	Sojourn time	Frequency
O_ACCEPTED	0ms 34s	2243
O_CANCELLED	851ms	3655
O_CREATED	59ms	7030
O_DECLINED	0ms	802
O_SELECTED	3s 851ms	7030
O_SENT	10d 2h	7030
O_SENTBACK	4d 8h	3454

Table 4. Sojourn times for the Offer states

- In between A_PARTLYSUBMITTED on the one hand and A_ACTIVATED, A_APPROVED, and A_REGISTERED on the other hand, the process can be canceled or declined. Before A_PARTLYSUBMITTED or after A_ACTIVATED, A_APPROVED, or A_REGISTERED this is not possible.
- Most time is spend in the A_FINALIZED state (more than 16 days), A_PREACCEPTED state (almost 3 days), and A_PARTLYSUBMITTED state (just over 2 days).
- The three parallel states (A_ACTIVATED, A_APPROVED, and A_REGISTERED) seem to be transient states, as they are (almost) immediately followed by another state. However, there is time involved in reaching the first of these events, and no other events follow them. If we would simply leave them, we would lose some information (the time to reach the first). Therefore, we choose to retain one (A_ACTIVATED) and ignore the other two in the remainder of this analysis.

4 Offer States

A quick inspection of the log $\mathcal{L}_O$ shows that there are 7 event classes, while the longest trace contains 30 events. Hence, this log does contain recurrent behavior. Therefore, for this log we opt for the set collection type with limit 1, which keeps the size of the resulting transition system at bay (only the last past event determines the state, hence there cannot be more states than event classes). The remainder of the settings are identical to the ones as used for the log $\mathcal{L}_A$. The left-hand side of Figure 2 shows the resulting transition system, whereas Table 4 and Table 5 show the corresponding timing information.

The frequencies from O_SELECTED to O_CANCELLED and from O_CANCELLED to O_CREATED are identical. This raises the hypothesis that triples (O_SELECTED, O_CANCELLED, O_CREATED), (O_SENT, O_CANCELLED, O_SELECTED), and (O_SENTBACK, O_CANCELLED, O_SELECTED) exist, but not, for example, (O_SENT, O_CANCELLED, O_CREATED) or (O_SELECTED, O_CANCELLED, O_SELECTED). To investigate this, we filtered the O_CANCELLED events out from the log and ran the Transition System Miner again with the same settings. The right-hand side of Figure 2 shows the resulting transition system. Indeed, there are no cases to go from O_SENT to O_CREATED or cases that go from O_SELECTED to itself. Therefore, we conclude that in the original transition system there are two types of canceled cases: Those that came from O_SELECTED and that will go to O_CREATED, and those that came from O_SENT or O_SENTBACK and that will go to O_SELECTED.

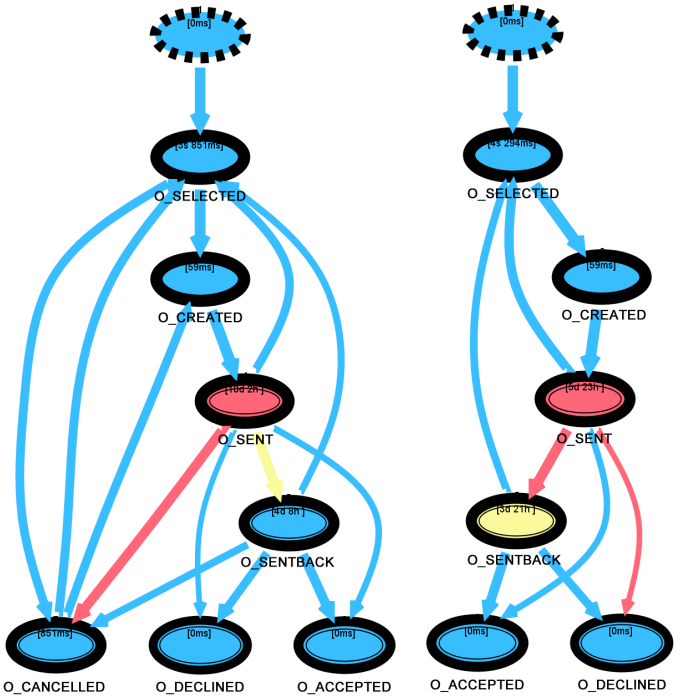

Fig. 2. Annotated transition systems for Offer states using 1-limited set abstraction.

The Offer process is started exactly as many times as state A_FINALIZED occurs: 5015 times This suggests that some relation exists between the O_FINALIZED state and the Offer process. Likewise, the Application state A_DECLINED 4 and the Offer state O_DECLINED both occur 802 times, and seem related. Although the match is not 100%, the Application state A_ACTIVATED (Remember that we decided to ignore the A_APPROVED and A_REGISTERED events earlier on) and the Offer state O_ACCEPTED seem related (2243 vs. 2246). Therefore, we have filtered in these events from the log $\mathcal{L}$, which results in the log $\mathcal{L}_{AO}$, and have mined a transition system from the resulting log using the following setting:

- Use *Activity* classifier for past events, no classifier for future events.
- Use unlimited bag abstraction (to be abel to detect parallelism with ease).
- Use all event classes.
- Use all transition labels.
- Use no post-mining operations.

Figure 3 shows the resulting transition system, which shows that the first O_SELECTED event occurs in parallel with A_FINALIZED and that A_ACTIVATED occurs in parallel with O_ACCEPTED.

In the end, we can conclude the following for the Offer states:

- The process starts with O_SELECTED.
- From the 5015 offers, 2243 were accepted, 802 were declined, 1640 were canceled, and 330 were abandoned (241 in the O_SENT state and 89 in the O_SENTBACK state).

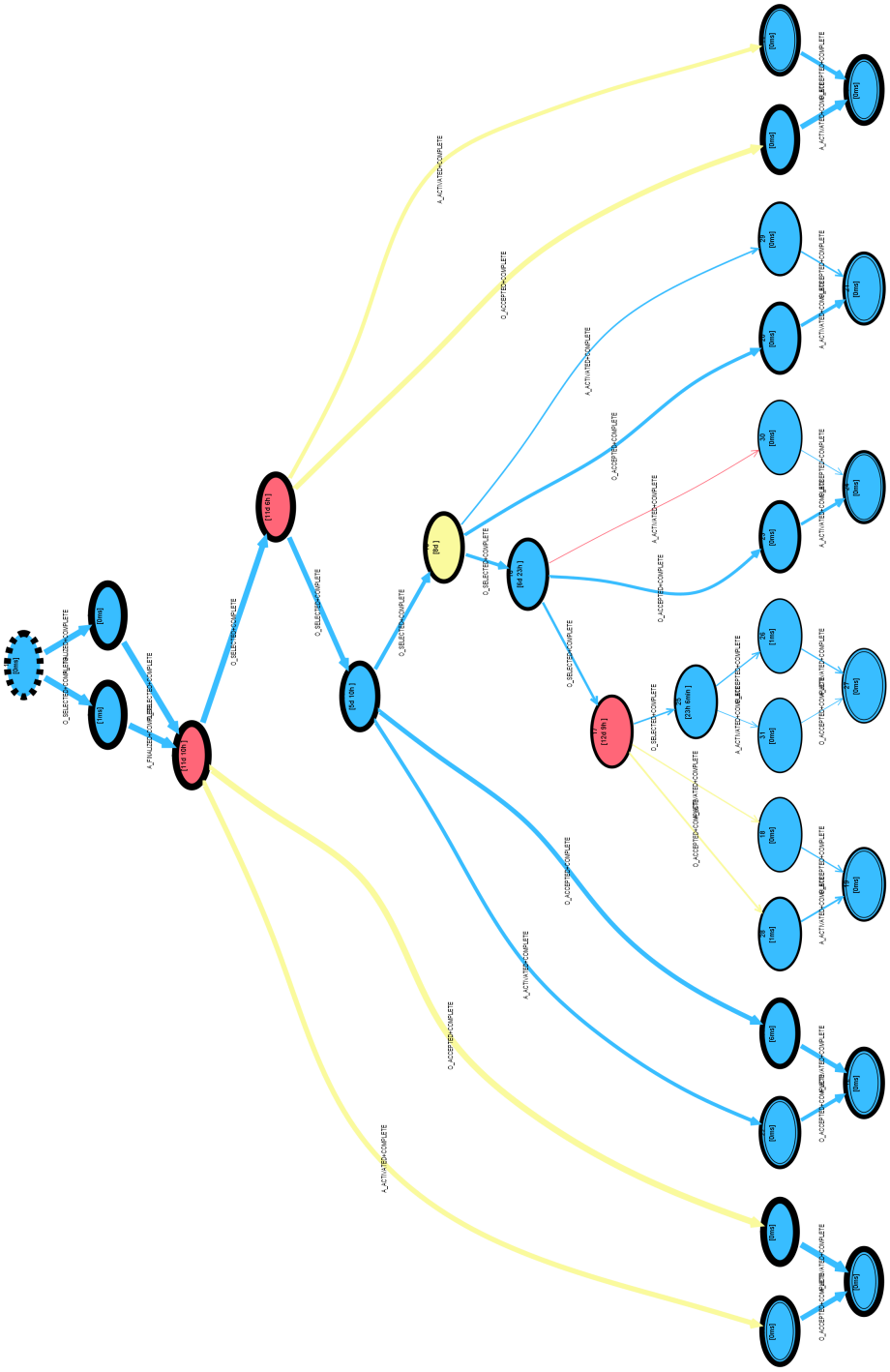

Fig. 3. Annotated transition systems for some Application and Offer states using unlimited multiset abstraction.

From State	To State	Duration	Frequency
	O_SELECTED	0ms	5015
O_CANCELLED	O_CREATED	3s 196ms	974
O_CANCELLED	O_SELECTED	0 ms	1041
O_CREATED	O_SENT	59ms	7030
O_SELECTED	O_CANCELLED	0ms	974
O_SELECTED	O_CREATED	4s 470ms	6056
O_SENT	O_ACCEPTED	4h 8min	130
O_SENT	O_CANCELLED	14d 4h	2373
O_SENT	O_DECLINED	7d 16h	54
O_SENT	O_SELECTED	5d 2h	778
O_SENT	O_SENTBACK	9d 14h	3454
O_SENTBACK	O_ACCEPTED	4h 6h	2113
O_SENTBACK	O_CANCELLED	7d 2h	308
O_SENTBACK	O_DECLINED	3d 18h	748
O_SENTBACK	O_SELECTED	4d 18h	196

Table 5. Sojourn times for the Offer transitions

- The first offers that is abandoned abandoned already in week 4.
- The offer can be canceled as long as it is in the O_SELECTED, O_SENT, or O_SENTBACK state. It is possible to follow up on a canceled offer, as this happens 2115 times.
- Most time is spend in the O_SENT state (more than 10 days) and the O_SENTBACK state (more than 4 days).
- If canceled from the O_SELECTED state, the next event will be a create event.
- If canceled from the O_SENT or O_SENTBACK states, the next event will be a O_SELECTED event.
- The Offer process starts when the Application process reaches the A_FINALIZED state.
- If the Offer process results in O_DECLINED, then the Application process results in A_DECLINED 4, and v.v.
- If the Offer process results in O_ACCEPTED, then the Application process results in A_ACTIVATED. For 2243 out of 2246 cases, this also holds in the opposite direction.

5 Work Item States

Like with $\mathcal{L}_O$, a quick inspection of $\mathcal{L}_W$ leads to the conclusion that the Work Item process contains recurrent behavior as well. Therefore, we use the same settings for mining the log $\mathcal{L}_W$ as for the log $\mathcal{L}_O$. However, as mentioned in Section 2, the Work Item events have three possible values for the lifecycle transition: SCHEDULE, START, and COMPLETE. To keep the result as simple as possible, we restricted ourselves to the SCHEDULE events for the time being. Hence, we first filtered in all SCHEDULE events from the log $\mathcal{L}_W$, and ran the Transition System Miner. The left-hand side of Figure 4 shows the resulting transition system, Table 6 and Table 7 shows the corresponding timing information.

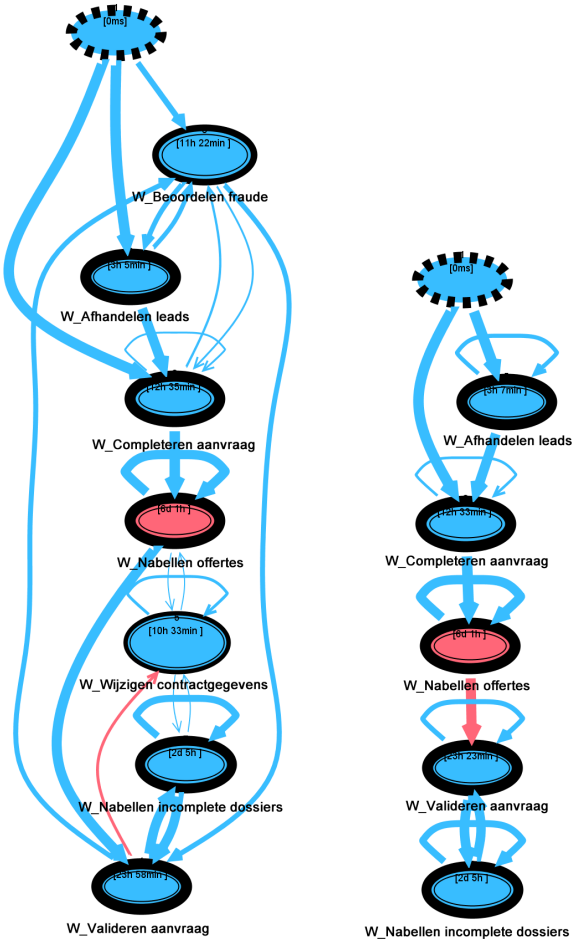

Fig. 4. Annotated transition systems for Work Item states using 1-limited set abstraction.

The transition system shows that the majority of cases first does an optional W_AFHANDELEN LEADS, then W_COMPLETEREN AANVRAAG, followed by a number of W_NABELLEN OFFERTEs, and a series of W_VALIDEREN AANVRAAG and W_NABELLEN INCOMPLETE DOSSIERS. The state W_WIJZIGEN CONTRACTGEGEVENS and W_BEOORDELEN FRAUDE seem to be infrequent (12 and 124 cases), where the later seems to be reachable from a lot of other states. If we filter these infrequent events out, we obtain the transition system at the right-hand side of Figure 4. Note how well-structured this transition system is. Apparently, the states W_WIJZIGEN CONTRACTGEGEVENS and W_BEOORDELEN FRAUDE are intermediate states that always lead back to the original states.

Next, we will try to link the Work Item events to the Application events. For this reason, we create a new log $\mathcal{L}_{AW}$ from the log $\mathcal{L}$ with all Application events

State	Sojourn time	Frequency
W_AFHANDELEN LEADS	3h 5min	4771
W_BEOORDELEN FRAUDE	11h 22min	124
W_COMPLETEREN AANVRAAG	12h 35min	7371
W_NABELLEN INCOMPLETE DOSSIERS	2d 5h	2383
W_NABELLEN OFFERTES	6d 1h	6634
W_VALIDEREN AANVRAAG	23h 58 min	5023
W_WIJZIGEN CONTRACTGEGEVENS	10h 33min	12

Table 6. Sojourn times for the Work Item states

From State	To State	Duration	Frequency
	W_BEOORDELEN FRAUDE	0ms	67
	W_AFHANDELEN LEADS	0ms	4739
	W_COMPLETEREN AANVRAAG	0ms	4852
W_AFHANDELEN LEADS	W_BEOORDELEN FRAUDE	3h 58min	21
W_AFHANDELEN LEADS	W_COMPLETEREN AANVRAAG	5h 49min	2515
W_BEOORDELEN FRAUDE	W_AFHANDELEN LEADS	1d 16h	32
W_BEOORDELEN FRAUDE	W_COMPLETEREN AANVRAAG	9h 16min	2
W_BEOORDELEN FRAUDE	W_VALIDEREN AANVRAAG	2h 32min	33
W_COMPLETEREN AANVRAAG	W_BEOORDELEN FRAUDE	5d 21h	3
W_COMPLETEREN AANVRAAG	W_COMPLETEREN AANVRAAG	55 min 2s	2
W_COMPLETEREN AANVRAAG	W_NABELLEN OFFERTES	18h 25min	5015
W_NABELLEN INCOMPLETE D.	W_NABELLEN INCOMPLETE D.	1d 15h	195
W_NABELLEN INCOMPLETE D.	W_VALIDEREN AANVRAAG	2d 20h	1736
W_NABELLEN INCOMPLETE D.	W_WIJZIGEN CONTRACTG.	14d 3h	1
W_NABELLEN OFFERTES	W_NABELLEN OFFERTES	5d 3h	1618
W_NABELLEN OFFERTES	W_VALIDEREN AANVRAAG	9d 19h	3254
W_NABELLEN OFFERTES	W_WIJZIGEN CONTRACTG.	22h 38min	1
W_VALIDEREN AANVRAAG	W_BEOORDELEN FRAUDE	2d 16h	33
W_VALIDEREN AANVRAAG	W_NABELLEN INCOMPLETE D.	2d 4h	2187
W_VALIDEREN AANVRAAG	W_WIJZIGEN CONTRACTG.	25d 21h	5
W_WIJZIGEN CONTRACTG.	W_NABELLEN INCOMPLETE D.	1d 4h	1
W_WIJZIGEN CONTRACTG.	W_NABELLEN OFFERTES	4d 1h	1
W_WIJZIGEN CONTRACTG.	W_WIJZIGEN CONTRACTG.	19s 982ms	5

Table 7. Sojourn times for the Work Item transitions

(except A_APPROVED and A_REGISTERED as explained earlier) and frequent SCHEDULE Work Item events filtered in. Figure 5 shows the resulting transition system. From this transition system, we can draw the following conclusions:

– W_AFHANDELEN LEADS is typically preceded by A_PARTLYSUBMITTED and is typically followed by either A_PREACCEPTED or A_CANCELLED.

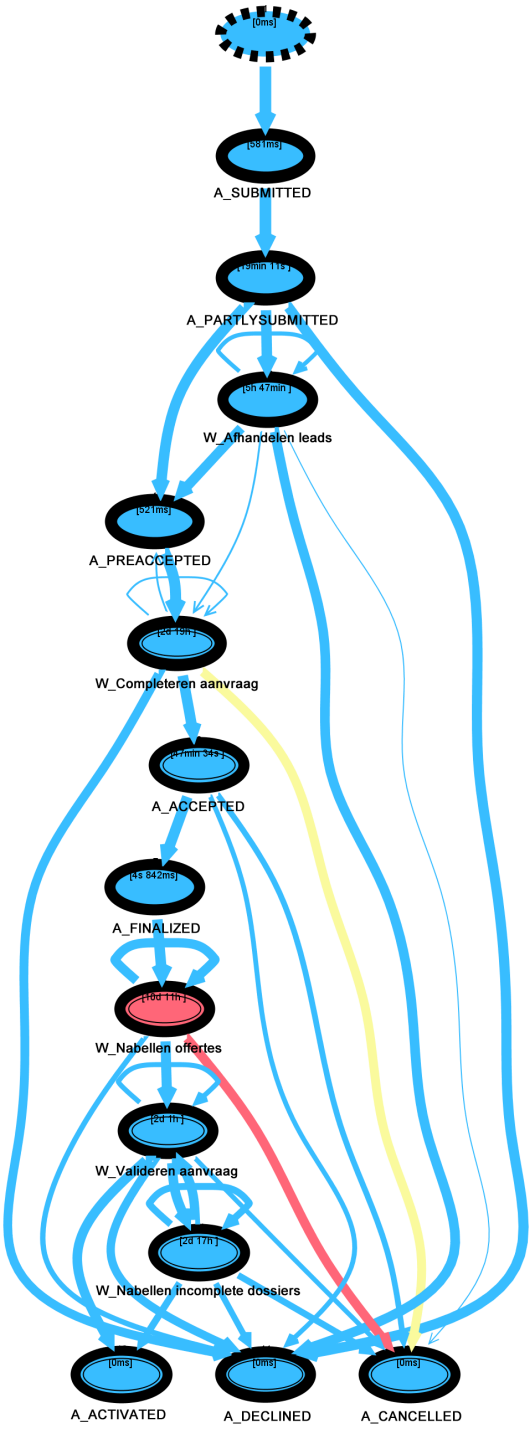

Fig. 5. Annotated transition systems for Work Item and Application states using 1-limited set abstraction.

- W_COMPLETEREN AANVRAAG typically follows A_PREACCEPTED and is typically followed by A_ACCEPTED, A_DECLINED, or A_CANCELLED.
- A_FINALIZED is followed by a series of W_NABELLEN OFFERTES, which is followed by a series of W_VALIDEREN AANVRAAG and W_NABELLEN INCOMPLETE DOSSIERS. These three W_ events are followed in the end by either A_ACTIVATED, A_DECLINED, or A_CANCELLED.
- The majority of applications get declined, and they get declined in an early stage of the process. Once an application has reached the A_FINALIZED stage, chances are that it will be activated.

6 Adding the Data Perspective

The log $\mathcal{L}$ comes with two specific trace attributes that might be of interest: AMOUNT_REQ and REG_DATA, where AMOUNT_REQ corresponds to the requested amount and REG_DATA corresponds to the date of registration of the application. Using the AMOUNT_REQ attribute, we will check whether the requested amount has any significant influence on the performance of the application process, using the REG_DATA attribute we will check whether the performance of the application process has improved significantly over time. To do so, we classify the possible values for both attributes into 5 buckets called 20, 40, 60, 80, and 100, where bucket X contains the applications for which the actual attribute value is in the interval $[l + (h - l) \times (X - 20)/100, l + (h - l) \times X/100)$, where l corresponds to the lowest value found in the log and h to the highest value. For sake of completeness, we mention that for AMOUNT_REQ, $l = 0$ (there is an application for which the requested amount is 0) and $h = 99999$, while for REG_DATA $l = 2011 - 10 - 01T00 : 38 : 44.546$ and $h = 2012 - 02 - 29T23 : 51 : 16.799$.

However, although the Transition System Miner can handle event attributes for identifying states, it cannot use trace attributes. For this reason, we have implemented a small plug-in that classifies both attributes as mentioned above, and that assigns the resulting classifications to the first event in the trace. The entire source of the plug-in can be found in Appendix A. The classified log for the Application event types is called $\mathcal{L}_A^c$ (we will ignore the other event types here as the Application events should already provide sufficient performance information). As we are mainly interested in performance information, we have filtered out all events except A_SUBMITTED, A_ACTIVATED, A_CANCELLED, and A_DECLINED, as these indicate the start and the end of the application process.

6.1 Requested amount

The left-hand side transition system in Figure 6 shows the result for the Application states, using the following settings for the Transition System Miner:

- Use *Activity* classifier for past events, no classifier for future events, select to use an unclassified attribute.
- Use 1-limited set abstraction.
- Use all event classes.
- Use unclassified requested amount bucket (AMOUNT_CLASS) attribute.
- Use no transition labels.
- Use no post-mining operations.

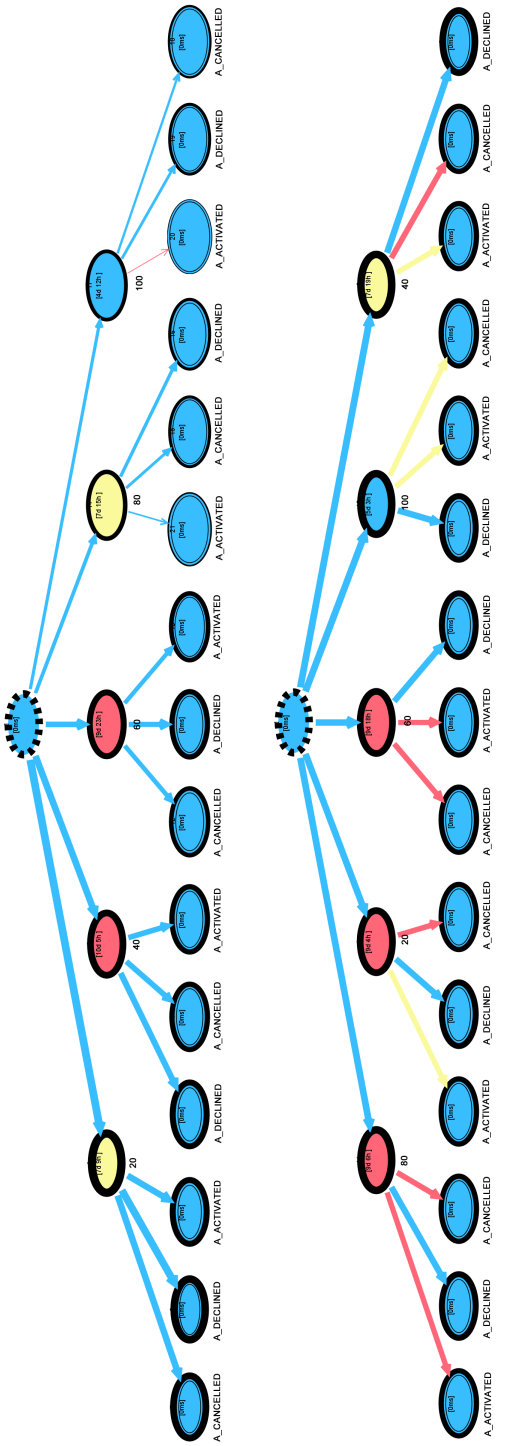

Fig. 6. Annotated transition systems for Application states classified by request amount (left-hand side) and registration date (right-hand side).

Bucket	Result	Elapsed time	Frequency
20	A_ACTIVATED	16d 10h	1579
20	A_CANCELLED	18d 11h	2018
20	A_DECLINED	1d 18h	6125
40	A_ACTIVATED	16d 19h	553
40	A_CANCELLED	19d 4h	605
40	A_DECLINED	2d 21h	1100
60	A_ACTIVATED	18d 18h	110
60	A_CANCELLED	19d 1h	159
60	A_DECLINED	4d 7h	364
80	A_ACTIVATED	17d 23h	3
80	A_CANCELLED	15d 5h	17
80	A_DECLINED	2d 10h	31
100	A_ACTIVATED	37d 1h	1
100	A_CANCELLED	6d 13h	8
100	A_DECLINED	1d 14h	15

Table 8. Elapsed times and frequencies per requested amount bucket.

Table 8 shows the elapsed times and the frequencies for the 5 buckets in combination with the result of the application. This Table shows that the majority of the applications are classified in the lower buckets, and that most of the applications get declined. Although the cardinalities of the higher buckets are relatively small, the Table does suggest that the chances to activate an application with a high amount (buckets 80 and 100) are relatively small when compared to the other buckets. Strange enough, though, the chances of having a very small (20) application activated seem smaller than the chances to have a moderate application (40 and 60) activated. Note, however, that Figure 6 suggests that these moderate applications typically take the most time, but this could be explained by the fact that activating an application takes typically more time than declining it. So, these moderate applications could take more time just because less get declined.

6.2 Log Period

We do the same for checking whether the process has improved over time, but we now take the unclassified registration date bucket (DATE_CLASS) attribute. The right-hand side transition system in Figure 6 shows the result, and Table 9 shows the elapsed times and the frequencies. This Table shows that the time to get an application activated or canceled suddenly drops in the last bucket (100).

7 Adding the Resource Perspective

Next to the Activity classifier, the log $\mathcal{L}$ contains a specific Resource classifier. As a result, the Transition System Miner can also be used to mine information from the resource perspective. As resources are typically associated with work items, we will

Bucket	Result	Elapsed time	Frequency
20	A_ACTIVATED	16d 22h	487
20	A_CANCELLED	18d 19h	529
20	A_DECLINED	2d 6h	1281
40	A_ACTIVATED	16d 21h	468
40	A_CANCELLED	19d 3h	643
40	A_DECLINED	1d 17h	1903
60	A_ACTIVATED	17d 23h	376
60	A_CANCELLED	21d 4h	535
60	A_DECLINED	2d 4h	1209
80	A_ACTIVATED	17d 21h	462
80	A_CANCELLED	19d 22h	612
80	A_DECLINED	2d 21h	1534
100	A_ACTIVATED	13d 18h	453
100	A_CANCELLED	13d 2h	488
100	A_DECLINED	1d 16h	1708

Table 9. Elapsed times and frequencies per registration date bucket.

limit ourselves to the log $\mathcal{L}_W$ here, and we will also limit ourselves to the W_NABELLEN OFFERTES START and COMPLETE events, as this activity takes the most time in the entire process. Therefore, we have filtered the W_NABELLEN OFFERTES events in and the SCHEDULE events out from the log $\mathcal{L}_W$, and obtained the log $\mathcal{L}_W^r$.

However, this log contains 53 resources. As displaying so many resources using a transition system is not a good idea, we have limited ourselves to the 5 most-frequent resources, which together cover 30% of all events. Figure 7 shows the resulting transition system, using the following settings for the Transition System Miner:

- Use *Activity* and *Resource* classifiers for past events, no classifier for future events.
- Use 1-limited set abstraction.
- Use 30 as threshold, as a result the 5 most-frequent resources are selected.
- Use no transition labels.
- Use no post-mining operation.

This Figure requires some explanation, as it contains START NONE and COMPLETE NONE states, but also START EMPTY and COMPLETE EMPTY states. The START NONE and COMPLETE NONE cases are a result of the fact that we have not selected all resources. As a result, if another resource starts an application, then there will be a START and/or COMPLETE lifecycle transition for the W_NABELLEN OFFERTES events, but there might not yet be a selected resource. This results in the START NONE and COMPLETE NONE states, as a latest selected resource does not exist yet. Please note that these states cannot be reached from the remainder of the transition system, which shows that this is indeed an initialization issue. In contrast, the START EMPTY and COMPLETE EMPTY states are a result of the fact that some events in the log do not have an associated resource, in which case an empty resource will be used. As this empty resource is the most-frequent resource, it appears in our list of 5 most-frequent resources, and is therefore a selected resource, which results in the START EMPTY and COMPLETE EMPTY states.

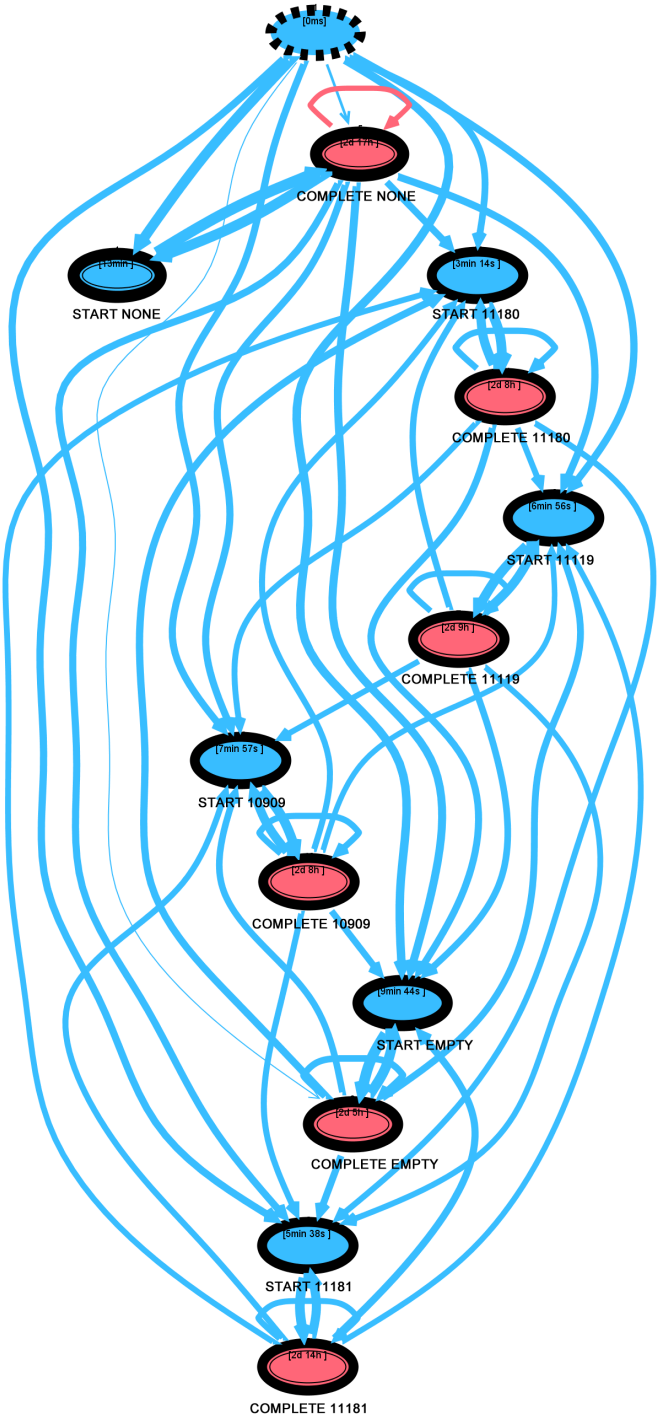

Fig. 7. Annotated transition systems for W_NABELLEN OFFERTE states for the 5 most-frequent resources.

Resource	Duration	Frequency
EMPTY	9min 44s	5364
10909	7min 57s	2061
11119	6min 56s	2315
11180	3min 14s	2423
11181	5min 38s	2327

Table 10. Duration from START to COMPLETE for the 5 most-frequent resources.

Note that the COMPLETE states are recurrent, while the START states can only be reached through the corresponding COMPLETE state. Apparently, it is possible to complete a started work item multiple times. For sake of curiosity, we decided to check the cases out for which this happens. To do so, we selected all the self-loop transitions as shown in Figure 7 in the mined transition system, and selected to filter the log on any of these transitions (threshold = 1). As a result, any application that contains a transition from a COMPLETE state to that same state for any of the 5 most-frequent resources will be filtered in. The resulting log contains 566 applications, and a similar transition system for the same resources was obtained. The remaining time from the initial state of this transition system to an accept state is on average $30d7h$ with a frequency of 566, whereas this was $11d9h$ with a frequency of 5011 for the transition system as showed in Figure 7. Therefore, we conclude that applications where any of the 5 most-frequent resources does a recurrent COMPLETE on the W_NABELLEN OFFERTES activity take considerably longer than applications for which this is not the case.

It seems that work is handed over (possibly through a number of unselected resources) between these resources. Therefore, we conclude that an application does not have a single resource as case manager. Possibly, the application requestor is called by different resources for the same application.

Table 10 shows the duration a resource requires on average to reach a COMPLETE state from the corresponding START state. Apparently, the EMPTY resource is slow, whereas the 11180 resource is quick when it comes down to completing the W_NABELLEN OFFERTE activity.

8 Conclusions

Using the Transition System Miner as available in ProM 6, we have investigated the control-flow perspective, the data perspective, and the resource perspective of the process that underlies the event log provided for the BPI Challenge 2012.

For the control-flow perspective, we discovered several transition systems that explain the underlying process, which seems to be nicely structured. For the Application events, we concluded that they occur in a certain order (see Figure 1), that the events A_ACTIVATED, A_APPROVED, and A_REGISTERED occur in parallel, and that the most time is spend in the A_FINALIZED state. For the Offer events, we concluded that they also occur in a certain order (see Figure 2), and that the successor of the O_CANCELLED state depends on its predecessor: If the predecessor was O_SELECTED, then the successor will be O_CREATED, else the successor will be O_SELECTED. For the combination of Application and Offer events, we concluded that the A_ACTIVATED events occur in parallel with the O_ACCEPTED events, and that the A_FINALIZED events occur in parallel

with the O_SELECTED events, which link both Figures nicely together. For the Work Item events, we concluded that, except for W_BEOORDELEN FRAUDE which can occur at almost any moment, they also occur in a certain order (see the right-hand side of Figure 4). For the combination of Application and Work Item events, we concluded that W_AFHANDELEN LEADS is typically preceded by A_SUBMITTED and typically followed by either A_PREACCEPTED or A_DECLINED, that W_COMPLETEREN AANVRAAG is typically preceded by A_PREACCEPTED and typically followed by either A_ACCEPTED or A_DECLINED, and that the other W_ events are typically preceded by A_FINALIZED and followed by either A_ACTIVATED, A_DECLINED, or A_CANCELLED.

For the data perspective, we have obtained transition systems for the Application events and have extended these with 5 buckets for the two trace attributes that were provided: AMOUNT_REQ (the requested amount for the application) and REG_DATA (the registration date for the application). Based on the minimal and maximal values as found for these attributes in the log, the entire range (from minimal to maximal) was split up into these 5 buckets, where bucket 20 corresponds to the lowest 20% in this range, 40 to the next 20%, etc. For the requested amounts, we concluded that the vast majority of the applications were for small amounts, and that only exceptionally a very high amount was requested. We could not conclude that more time was spend on applications that involved a higher requested amount, but we did note that the buckets 40 and 60 contained a relatively high number of activated cases (1 out of 4, 1 out of 5), whereas the 80 and 100 buckets contained a relatively small number of activated cases (1 out of 10, 1 out of 15). For the registration date, we concluded that the time to handle an application was especially low in the 100 bucket, that is, at the end of the logged period.

For the resource perspective, we have obtained transition systems from the Work Item events (as these were assumed to be related to resources), and have investigated behavior of the 5 most-frequent resources related to the W_NABELLEN OFFERTES Work Item. From this, we concluded that no case manager is associated to an application in the geven process, and that the 5 selected resources all handover work (possibly through a number of unselected resources) to each other. We also concluded that for some of the Work Item events no resource was specified, and that this EMPTY resource was actually the most-frequent resource. Finally, we concluded that the selected resources all could redo the completion of the Work item without restarting it, and that the applications in which this redoing occur typically take more time than the applications in which this does not occur.

To get to these conclusions, we mainly used the Simple Log Filter, the Transition System Miner, and the Transition System Analyzer plug-ins as they are implemented in ProM 6. This shows that this set of plug-ins is very versatile, and that many different types of results can be obtained using them. The Simple Log Filter was used to obtain a log that contains the events one is interested in. The Transition System Miner was used to create a transition system from an event log, where basically any combination of attributes present in the log can be used to identify the states in the transition system. Although the Miner can handle unclassified attributes, it is much more powerful if classified attributes are used, which emphasizes the need for good classifiers in an event log. The Transition System Analyzer was used to extend the mined transition system with timing and frequency data as aggregated from the event log.

References

1. Dongen, B.F.v.: BPI challenge 2012. Dataset. http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f (2012)
2. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Prom 6: The process mining toolkit. In Rosa, M.L., ed.: Proc. of BPM Demonstration Track 2010. Volume 615 of CEUR Workshop Proceedings., Hoboken, USA, CEUR-WS.org (September 2010) 34–39
3. Aalst, W.M.P.v.d., Rubin, V., Verbeek, H.M.W., Dongen, B.F.v., Kindler, E., Günther, C.W.: Process mining: A two-step approach to balance between underfitting and overfitting. Software and Systems Modeling (SoSyM) **9**(1) (2010) 87–111

A Classification plug-in

This Appendix shows the source code for the plug-in that was used to classify the applications based on the requested amount and registration date. For both, 5 buckets are created (20, 40, 60, 80, and 100), and the resulting classification is added to the first event of the corresponding trace.

```
package org.processmining.plugins.bpic2012;

import java.util.HashMap;
import java.util.Map;

import org.deckfour.xes.factory.XFactoryRegistry;
import org.deckfour.xes.info.XLogInfoFactory;
import org.deckfour.xes.model.XAttribute;
import org.deckfour.xes.model.XAttributeLiteral;
import org.deckfour.xes.model.XAttributeTimestamp;
import org.deckfour.xes.model.XEvent;
import org.deckfour.xes.model.XLog;
import org.deckfour.xes.model.XTrace;
import org.processmining.contexts.uitopia.annotations.
    UITopiaVariant;
import org.processmining.framework.plugin.PluginContext;
import org.processmining.framework.plugin.annotations.Plugin;
import org.processmining.plugins.log.logfilters.LogFilter;
import org.processmining.plugins.log.logfilters.
    LogFilterException;
import org.processmining.plugins.log.logfilters.XEventEditor;

public class ClassifyFirstEvent {
  private static int minAmount, maxAmount;
  private static long minTime, maxTime;
  Map<XEvent, Integer> amountMap = new HashMap<XEvent, Integer
      >();
  Map<XEvent, Long> timeMap = new HashMap<XEvent, Long>();
```

```
@Plugin(name = "Classify (BPIC 2012)", parameterLabels = { "
    Log" }, returnLabels = { "Log (classified)" },
    returnTypes = { XLog.class })
@UITopiaVariant(affiliation = UITopiaVariant.EHV, author = "
    H.M.W. Verbeek", email = "h.m.w.verbeek@tue.nl")
public XLog classify(PluginContext context, XLog log) throws
     LogFilterException {
  minAmount = -1;
  maxAmount = -1;
  minTime = -1;
  maxTime = -1;
  for (XTrace trace : log) {
    XAttribute amountAttr = trace.getAttributes().get("
        AMOUNT_REQ");
    if (amountAttr instanceof XAttributeLiteral) {
      int amountReq = Integer.parseInt(((XAttributeLiteral)
          amountAttr).getValue());
      if (minAmount < 0 || minAmount > amountReq) {
        minAmount = amountReq;
      }
      if (maxAmount < 0 || maxAmount < amountReq) {
        maxAmount = amountReq;
      }
      amountMap.put(trace.get(0), amountReq);
    }
    XAttribute timeAttr = trace.getAttributes().get("
        REG_DATE");
    if (timeAttr instanceof XAttributeTimestamp) {
      long value = ((XAttributeTimestamp) timeAttr).getValue
          ().getTime();
      if (minTime < 0 || minTime > value) {
        minTime = value;
      }
      if (maxTime < 0 || maxTime < value) {
        maxTime = value;
      }
      timeMap.put(trace.get(0), value);
    }
  }
  return LogFilter.filter(context.getProgress(), 100, log,
      XLogInfoFactory.createLogInfo(log),
      new XEventEditor() {
        public XEvent editEvent(XEvent event) {
          XEvent editedEvent = (XEvent) event.clone();
          if (amountMap.containsKey(event)) {
            int amount = amountMap.get(event);
            XAttribute attr;
            if (amount < minAmount + (maxAmount - minAmount)
                / 5) {
```

```
      attr = XFactoryRegistry.instance().
          currentDefault()
          .createAttributeLiteral("AMOUNT_CLASS", "
              20", null);
    } else if (amount < minAmount + 2 * (maxAmount -
        minAmount) / 5) {
      attr = XFactoryRegistry.instance().
          currentDefault()
          .createAttributeLiteral("AMOUNT_CLASS", "
              40", null);
    } else if (amount < minAmount + 3 * (maxAmount -
        minAmount) / 5) {
      attr = XFactoryRegistry.instance().
          currentDefault()
          .createAttributeLiteral("AMOUNT_CLASS", "
              60", null);
    } else if (amount < minAmount + 4 * (maxAmount -
        minAmount) / 5) {
      attr = XFactoryRegistry.instance().
          currentDefault()
          .createAttributeLiteral("AMOUNT_CLASS", "
              80", null);
    } else {
      attr = XFactoryRegistry.instance().
          currentDefault()
          .createAttributeLiteral("AMOUNT_CLASS", "
              100", null);
    }
    editedEvent.getAttributes().put("AMOUNT_CLASS",
        attr);
  }
  if (timeMap.containsKey(event)) {
    long time = timeMap.get(event);
    XAttribute attr;
    if (time < minTime + (maxTime - minTime) / 5) {
      attr = XFactoryRegistry.instance().
          currentDefault()
          .createAttributeLiteral("DATE_CLASS", "20"
              , null);
    } else if (time < minTime + 2 * (maxTime -
        minTime) / 5) {
      attr = XFactoryRegistry.instance().
          currentDefault()
          .createAttributeLiteral("DATE_CLASS", "40"
              , null);
    } else if (time < minTime + 3 * (maxTime -
        minTime) / 5) {
      attr = XFactoryRegistry.instance().
          currentDefault()
```

```
                    .createAttributeLiteral("DATE_CLASS", "60"
                        , null);
                } else if (time < minTime + 4 * (maxTime -
                    minTime) / 5) {
                  attr = XFactoryRegistry.instance().
                      currentDefault()
                      .createAttributeLiteral("DATE_CLASS", "80"
                          , null);
                } else {
                  attr = XFactoryRegistry.instance().
                      currentDefault()
                      .createAttributeLiteral("DATE_CLASS", "100
                          ", null);
                }
                editedEvent.getAttributes().put("DATE_CLASS",
                    attr);
              }
              return editedEvent;
            }

          });
  }
}
```