Petra: A Tool for Analysing a Process Family

D.M.M. Schunselaar*, H.M.W. Verbeek*, W.M.P. van der Aalst*, and H.A. Reijers*

Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
{d.m.m.schunselaar, h.m.w.verbeek, w.m.p.v.d.aalst, h.a.reijers}@tue.nl

Abstract. When looking for the best model, simulation is often used for "what-if" analysis. The properties of a modelled process are explored by repeatedly executing the model. Based on the outcomes, parts of the model may be manually modified to improve the process. This is iteratively done to obtain the model best suited to a user's requirements. In most cases, this is a labour-intensive and time-consuming task. To improve on the state of the art, we propose a framework where the user defines a *space of possible process models* with the use of a configurable process model. A configurable process model is a compact representation of a family of process models, i.e., a set of process models related to each other. Within the framework, different tools can be used to automatically compute characteristics of a model. We show that, when used on data from multiple real-life models, our framework can find better models in an automated way.

1 Introduction

Within the CoSeLoG project, we are cooperating with 10 Dutch municipalities. Each of these municipalities has to provide the same set of services to their citizens, but each may do so in its own way [1]. This "couleur locale" justifies that there may be different solutions to realise a particular process. The union of local variations for these services spans a solution space containing the process models currently selected by municipalities (and possibly more). When a municipality wants to improve their process model, they can use the solution space to find a better solution (process model) to replace their process model. In Fig. 1, we illustrate the solution space spanned by interpolating between the different process models from the municipalities A, B, and C. In [1], we have shown how to obtain this solution space. This initial solution space lacks the explicit information which of the models is the most desired one for an organisation (in Fig. 1 indicated by "?/?"). In this paper, we present *Petra*: A generic tool to explore a space of possible models by repeatedly analysing elements of the space and supporting the collection and comparison of analysis results. Using *Petra*, we transform the initial solution space to a solution space with explicit information

* This research has been carried out as part of the Configurable Services for Local Governments (CoSeLoG) project (http://www.win.tue.nl/coselog/).

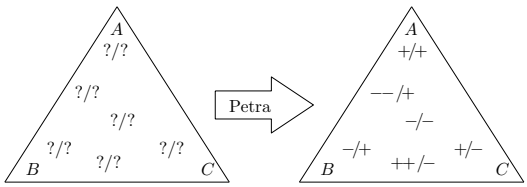

Fig. 1: Given a space of models spanned by a family of process models, we can compute characteristics (2 in this example) for all the process models. These characteristics can be used to find the "best" model.

about various aspects of the model, e.g., sojourn time, cost, and complexity (in Fig. 1 indicated with "+/-", "+/+", etc.).

Some approaches exist for the problem at hand. Unfortunately, these approaches are either only existent on paper, or they are limited to a single tool capable of computing a predetermined set of characteristics. Therefore, we introduce *Petra*[1], which stands for "Process model based Extensible Toolset for Redesign and Analysis". *Petra* is a generic and extensible framework. The solution space it operates on is defined using configurable process models. The values obtained from different analysis techniques are stored using *properties*. By providing a generic interface, any analysis tool can be used in our framework by implementing this interface.

Configurable process models are a compact representation for a family of process models, i.e., a set of process models which are related to each other. Different members of the family can be obtained by selecting different elements (from a predetermined set of elements) to be removed. This selection is called a configuration. This notion of configurable process models subsumes adjusting the model with the use of XOR/OR gateways. For our configurable process models, we use block-structured process models called Process Trees [2], which are a generalisation of the formalism presented in [1]. Process Trees are specifically developed within our project and all developed analysis techniques have been designed for Process Trees.

The executable model obtained after configuring the configurable process model is a model from the aforementioned solution space which one would like to analyse. Using the aforementioned properties, we can annotate a process model with analysis results. Properties are generic and can encode any analysis result obtainable, e.g., sojourn time, costs, etc. Apart from analysis results, properties are also used to encode run-time characteristics of a process model, e.g., the arrival process of cases, work schedules of resources, and variable working speeds of resources.

Since any analysis result can be encoded and we do not want to limit the analysis power of *Petra*, we provide a generic tool interface. On this interface, we provide a configured process model with relevant properties. The analysis

[1] *Petra* is implemented as a ProM plugin http://www.processmining.org/

tools interacting with *Petra* are expected to, if required, make a transformation from our Process Trees to the tool specific formalism. Conversions already exist from Process Trees to classical Petri nets, BPMN, and YAWL for structural analysis. When the tool finishes its analysis, we expect it to return a Process Tree annotated with analysis results. We have used CPN Tools [3] as a first example of an analysis tool in the context of *Petra*. We have selected CPN Tools for 3 reasons; (1) it provides flexibility in defining the to-be-computed characteristics, (2) by using Coloured Petri Nets, we can easily encode the more advanced insights in resource behaviour, and (3) through Access/CPN [4], it is possible to simulate all the models without human involvement.

Apart from providing the *Petra* framework and its implementation, we also conducted a case study in which we applied our tool to the models from the municipalities involved in the CoSeLoG project. In this case study, we have taken models from two municipalities and transformed these into a configurable process model. For each of the configured models in the solution space, we automatically created a CPN model and simulated the CPN model to obtain time-related information. Afterwards, we have enriched the analysed models with the output of the simulation. This case study shows that our simulation models approach reality and that we can find better models using *Petra*.

The remainder of this paper is organised as follows: Sect. 2 presents related work. In Sect. 3, we present our high-level architecture, Process Trees, and the properties currently used within our framework. The transformation of a Process Tree with properties to a CPN model is presented in Sect. 4. After presenting the key concepts of our framework, we use a case study to demonstrate the applicability of *Petra* (Sect. 5). Finally, in Sect. 6, we present our conclusions and sketch venues for extensions and research. A technical report supporting this paper can be found in [2].

2 Related work

In this section, we will first elaborate on the techniques applicable to our problem setting. Afterwards, different techniques are discussed which reason on a configurable process model to obtain a process model most desired by the user. Finally, we briefly discuss the used formalism for *Petra*, as well as its limitations and benefits relative to other formalisms.

2.1 Applicable techniques for our problem setting

In [5], an approach is presented that is based on configurable process models. As far as we know, it is the approach most similar to ours. The configurable process models that are used by it are modelled in Protos [5]. By converting the configurable process model to a CPN model, the authors can use the same model to analyse multiple variants of the same process model. The main limitation of their approach is the fact that the focus is on the use of a single tool (CPN Tools) which results in a non-extensible set of analysis results. Furthermore, the

resource model employed is rather simplistic (see [6], for an overview on the importance of correctly modelling resources), i.e., all resources are available all of the time. There is also no support for data, i.e., exclusive choices are chosen randomly. Finally, with respect to determining soundness, state space analysis has to be employed.

Another approach related to our approach is presented in [7]. On the existing process model, process measures are computed. Afterwards, different applicable best practices are evaluated and applied to obtain new process models. Finally, on these process models, process measures are computed, the best-performing process models are kept, and the steps are repeated. In [8], an implementation is presented. However, the focus is on simulation and time-related information. Furthermore, a single tool is selected for the analysis, prohibiting to gather information not provided by this specific tool.

In [9], there is a direct dependency on process mining techniques to obtain the process model as-is, and on enrichment of the process model with information from the event log. After obtaining the enriched model, there is an iterative approach of finding the malfunctioning parts of the process model, selecting transformation from a database to be applied, generating process models for the different redesign possibilities. The generated process models are stored, and, if required, the aforementioned steps can be re-executed to change another part of the process model. From all the generated process models, the best process model is selected and returned to the user. However, it is unclear how the database of redesigns is obtained, and it has not been implemented.

2.2 Configurable process models

In [10], a questionnaire-based approach is presented to deduce on a high level the requirements a user poses on the best instantiation of a configurable process model. This, in combination with functional requirements, results in an instantiated process model closest to the user's requirements. This approach does not give any performance information, but it can be used beforehand to limit the to-be-analysed solution space.

The `Provop` framework contains context-based configuration of the configurable process model [11]. Within the `Provop` framework, there are so-called context rules which limit the configuration possibilities by including contextual information. These context rules specify, for instance, that if the user requests high quality, then certain other activities have to be included in the process model. As with the approach in [10], the focus of this approach is not on collecting performance information. Yet, it can be used to limit the solution space.

2.3 Process Trees and properties

Configurable process models have been defined for different modelling formalisms, e.g., configurable EPC [12,13], configurable YAWL [14], configurable BPEL [14], CoSeNets [1], and Process Trees [2]. The first 3 formalism are more expressive

than the latter two, i.e., the CoSeNets and Process Trees are a subclass of Free-Choice Petri nets. However, with the first 3, explicit attention has to be paid to the soundness [1] of the configured model. Furthermore, CoSeNets only focus on control flow. Therefore, we use Process Trees as our formalism.

Process Trees are related to the RPST [15] in the sense that both are block-structured. However, the RPST is used to convert non-block structured models into block-structured models and focusses on the control-flow perspective. Furthermore, RPST can have so-called rigids which are non block-structured fragments of the process model. These are not present in Process Trees.

Efforts have been made on enriching BPMN models with simulation information by the *WfMC* standard *BPSim* [16]. However, we allow, amongst others, for a richer resource model through supporting arousal-based working speeds. Furthermore, *BPSim* is tailored towards simulation and thus abstracting from non-simulation related information, e.g., compliance. We would like to be able to encode this non-simulation related information as this might be of importance for different analysis techniques. The properties are related to and inspired by *BPSim* and a transformation from properties to *BPSim* has been made which works in conjunction with the L-SIM tool from Lanner[2].

3 Petra

In this section, we show the high-level architecture of *Petra*. We will also discuss the lefthand side of Fig. 2, i.e., the process model and the properties used.

3.1 Architecture

The architecture of *Petra*, including the use of our sample, analysis tool, is depicted in Fig. 2. In *Petra*, we have a family of process models defined by a configurable process model from which we want to select the "best". This is

[2] `http://www.lanner.com/en/l-sim.cfm`

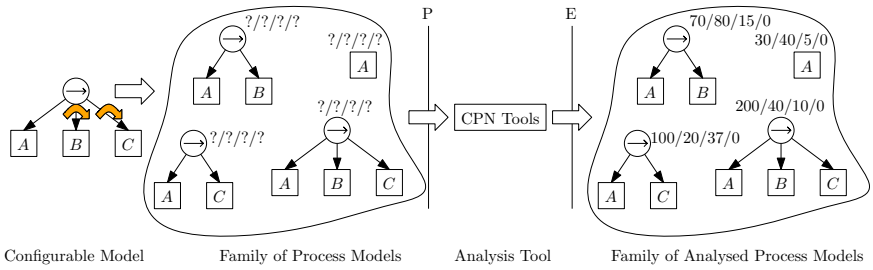

Fig. 2: In *Petra*, we start with a configurable process model which describes a family of process models. Each of these models is analysed resulting in a family of analysed process models.

subject to the exact requirements of the organisation. We extend every process model with relevant properties computed using different tools, e.g., sojourn time, queue time, costs, etc. After obtaining the relevant properties, we have a family of analysed models. At a later stage, the user can, via different views on this family of models, select the desired process model.

Petra consists of 3 key concepts: a family of process models, a set of properties, and a tool interface. We first elaborate on the properties and the tool interface. Afterwards, the Process Trees are discussed, and the family of process models last.

Properties Properties come in two different flavours; independent (facts), and dependent (deduced from facts). By not limiting the framework to a fixed set of properties, we keep our framework generic. Furthermore, properties can be part of every construct, e.g., resources, data, etc. In this paper, we focus on an often used quantitative measure namely time. But properties can also be used to encode for instance the model complexity.

Tool interface To the analysis tool employed, we offer a Process Tree annotated with properties on interface P. An analysis tool has to make a transformation from the Process Tree with properties, to the tool-specific formalism or use one of the currently available conversions, e.g., to classical Petri Nets or BPMN. Afterwards, if the tool has finished its computations, the output of the tool has to be transformed into dependent properties and the Process Tree should be enriched with this information (interface E). The properties offered on interface P can both be independent, and dependent. In our framework, we assume an incremental approach, i.e., tools may only modify or add dependent properties and are not allowed to remove (in)dependent properties. Since the amount of possible models can be exponential in the amount of configuration points, we require a tool to run automatically, i.e., without the need of human involvement. Interface P can also be used to query properties of a tool, e.g., properties the tool can compute (and for which nodes), and properties necessary for that.

3.2 Process Trees

Process Trees are block-structured process models, a subclass of Free-Choice Petri nets, in which each block has a single entry and a single exit. A Process Tree consists of 3 perspectives: control-flow, resource, and data. Next to this, we have two extra perspectives to encode contextual information namely: environment, and experiment. In Fig. 3, the different perspectives are depicted and their relation to each other. Each of the perspectives contains a set of configuration points with configuration options. In the remainder of this section, we deal with each of the perspectives and their properties, and show the configuration options.

Control-flow perspective The control-flow perspective consists of nodes, and directed edges between these nodes which denotes the order of execution. Nodes

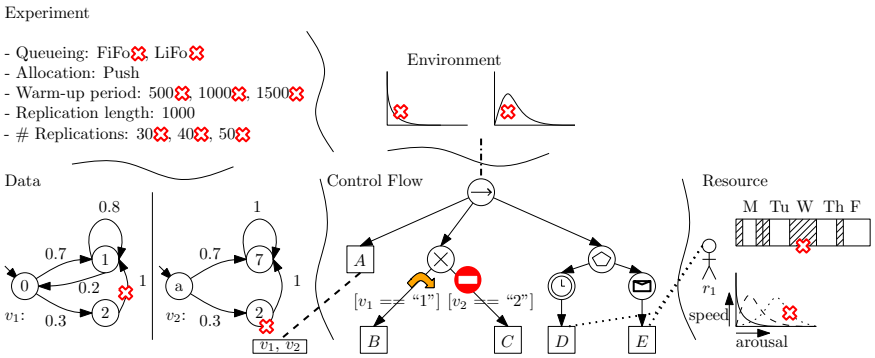

Fig. 3: The different perspectives in a Process Tree.

come in two different flavours: tasks and blocks. Tasks are the units of work which need to be executed, e.g., A in Fig. 3, and can be manual (with a resource) or automatic. Blocks specify the causal relationship between its children, e.g., the block with the arrow in Fig. 3 denotes a sequential execution. All the nodes are depicted in Fig. 4 with their Petri net semantics[3]. Note that, we use some of the notations for events in Petri nets from [17].

If there is an edge from a block to a node, then we say that this node is a child of that block. We have a total order on the outgoing edges as for most blocks the order has semantics, e.g., SEQ. In general, all nodes can have any number of children excepts for the EVENT (letter or clock), LOOPXOR, and LOOPDEF (see Fig. 4). Finally, the set of nodes and edges forms a connected Directed Acyclic Graph (DAG) with a single root node, i.e., a node without any incoming edges. Process Trees are encoded as a DAG to minimise duplication of behaviour and thus increasing maintainability.

There are three types of configuration options: (1) hiding, (2) blocking, and (3) substitution. Substitution entails the option to replace a part of the process model with a subprocess from a predetermined collection of subprocesses (see [1]). Hiding, which is shown in Fig. 5 with the curved arrow, entails the option to abstract from a part of the process model by substituting the subprocess with an automatic task, e.g., Fig. 5(b). Blocking, shown with a no-entry sign in Fig. 5, denotes the option to prevent the flow of entering a part of the process model, e.g., Fig. 5(a). Note that blocking has non-local effects, e.g., if a part of a sequence is blocked, then the entire sequence becomes blocked.

In Fig. 5, the space of models is depicted ((a)-(d)) using hiding and blocking. Figure 5(a) and (c), shows the configured model if we select blocking (note that we have removed the XOR as it is redundant), Fig. 5(b) and (c) shows the configured model if hiding is selected and finally, Fig. 5(d) shows the configured model when none of the configuration options is selected.

[3] Note that, for many nodes, Fig. 4 shows the semantics for the case with only two children. However, it is trivial to extend this to more children.

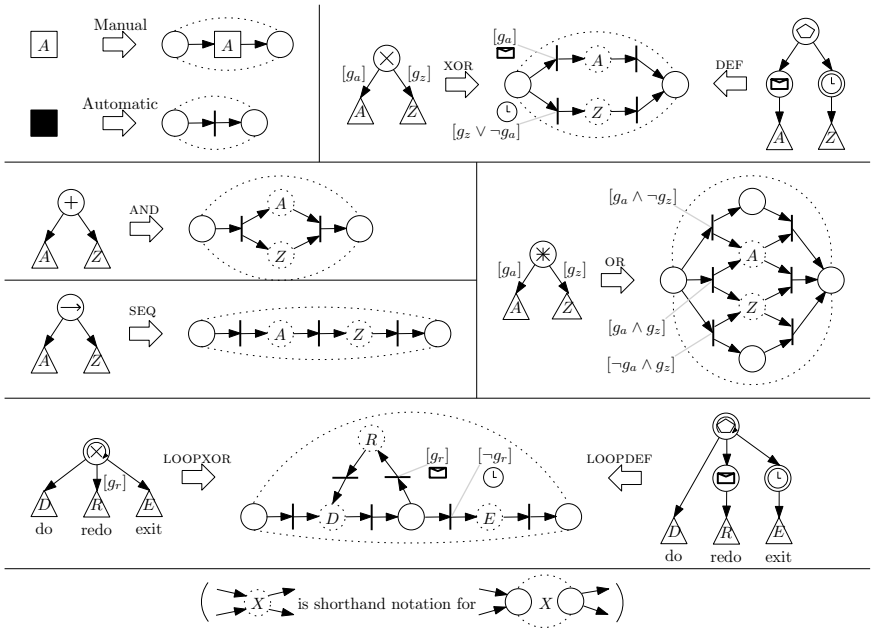

Fig. 4: The different nodes and their (abstract) Petri net semantics.

Data perspective The data perspective specifies which expressions are used for the outgoing edges of a choice (between "[" and "]"), which variables are read and written (line from A to v_1 and to v_2 in Fig. 3). Furthermore, we have variables encoded as Markov chains denoting which values a variable might take, with which probability it may take this value, and what the initial value is of a variable. For instance, in Fig. 3 (left-hand side), the variable v_1 has initial value "0", it may take the values "0", "1", and "2", and it changes value according to the probabilities on the edges.

Expressions reason over variables and values for those variables. They have the following operators: conjunction, disjunction, and negation. Furthermore, variables may be compared to values on equality and inequality.

On the outgoing edges of choice constructs, there is the option to select an expression from a set of expressions. For variables, we can change which nodes read/write this variable. Furthermore we have the following properties for variables: we have the option to change the initial values, which values may be taken, and whether an edge with a certain probability exists between two values. In Fig. 6, the different possibilities are shown for removing a potential value and removing an edge between two values for a variable. The family of data perspectives is spanned by the cartesian product of the options for the variables and expressions.

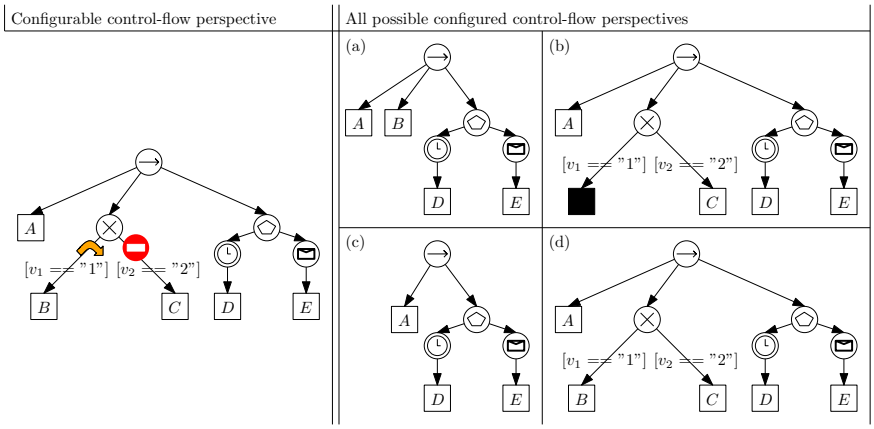

Fig. 5: An example family of control flows.

Resource perspective The resource perspective specifies which resource may execute which activity, e.g., in Fig. 3, we have a resource r_1 and she may execute D and E. Furthermore, the properties of the resource perspective specify (1) when a resource is available to work on the process (closely related and inspired by [6]), e.g., r_1 is available on Monday morning and evening, but not on Friday, and (2) how many resources there are of a particular kind. Finally, different working speeds can be specified based on the busyness of a resource, this to model effects such as the phenomenon described by the Yerkes-Dodson law of arousal [18][4]. For the work schedule, one can remove intervals in which the resource is available to the process (Fig. 7 at the top). The number of resources is selected from a predetermined set of values. Finally, the arousal based work speed offers the option to remove parts of the work speeds associated with the arousal levels (Fig. 7 at the bottom). The family of resource perspectives is spanned by the cartesian product of the options for the work schedule, arousal based work speed, and number of resources.

Environment perspective Currently, we only have the option to select the arrival process (a property) for the environment perspective. The arrival process specifies the distribution of arrivals of new cases to the process. Configuring the arrival process entails selecting a distribution describing the arrival of cases.

Experiment perspective The experiment perspective consists of the simulation property (Fig. 3 top left), which specifies the arguments for the simulation tool. We can select a queuing principle, e.g., FiFo, and whether push or pull allocation is used. Furthermore, we can set the warm-up period, replication length, and

[4] The Yerkes-Dodson law of arousal describes the correlation between the arousal (pressure) and the working speed of a person

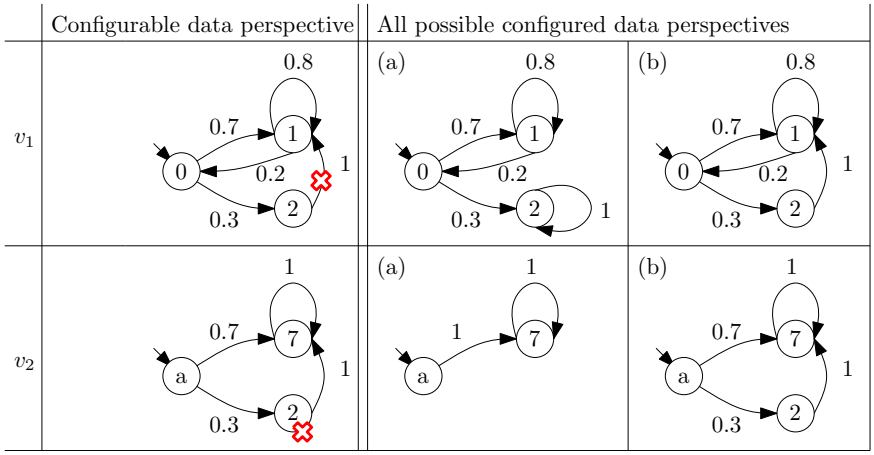

Fig. 6: Example families of Markov chains for variables v_1 and v_2.

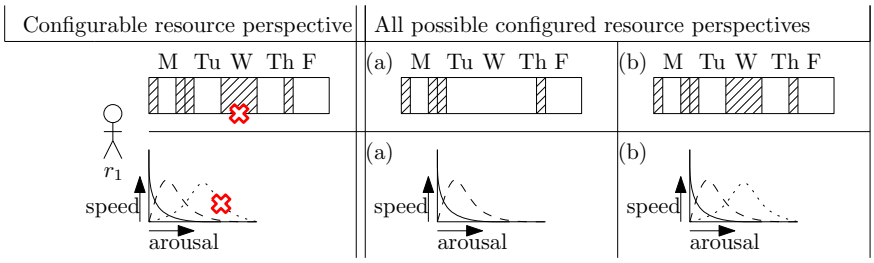

Fig. 7: Example families of work schedules and work speeds for resource r_1.

number of replications. For each of the arguments, a single value has to be selected from a predetermined set of values.

The properties inside of the experiment perspective are not entirely independent of the used tool, e.g., the simulation property uses concepts from the simulation domain. When analysis techniques from another domain is used, there should be a property with common concepts from that domain.

3.3 Solution space

The solution space of a Process Tree is spanned by the cartesian product of possible selections of configuration options for the various configuration points in the various perspectives. In case of Fig. 3, we have a total of $2304 = 2 \cdot 4 \cdot 4 \cdot 4 \cdot 18$ possible models, i.e., 2 possibilities for the environment perspective, 4 for the data perspective, 4 for the control-flow perspective, 4 for the resource perspective, and 18 for the experiment perspective. *Petra* is able to traverse this solution space automatically irrespective of the used properties.

4 Sample tool: CPN Tools

In this section, we elaborate on the transformation of a configured Process Tree to a CPN model. Since the implementation for controlling CPN Tools using Access/CPN and parsing the output is relatively straightforward [2], we do not elaborate on it here.

4.1 Transforming a Process Tree to a CPN model

Transforming a Process Tree to a CPN model is done along four perspectives: control-flow, data, resources, and environment; the experiment perspective consists mainly of controlling CPN Tools itself. Each of the four perspectives is treated separately. Modifications that only impact on a single perspective also only modify the CPN model in that perspective. Between different perspectives, we have introduced the notion of a controller such that the perspectives can communicate with each other. For example, the control-flow perspective uses the allocation controller to signal that a resource is necessary.

Data perspective The transformation of the expressions in the Process Tree is straightforward, i.e., we have a variable (v_{expr}) storing whether an expression evaluated to true of false. A transition guarded with the expression itself is in charge of updating v_{expr} based on the values of the variables in the expression and based on the previous version of the expression.

Modelling the variables themselves is a bit more involved. When a new case is started, the variable is initialised with the initial value. If a variable is written, we determine the new value of a variable. Most notable for the variable is the fact that we employ two different views on the variables: a control-flow view, and a guard view. We made the distinction because the different views serve two different purposes. For the control-flow view, it is irrelevant what the value of a variable is but it is important to know whether an update has been performed. For the guard view, it is important to know the exact value of a variable as these are used to evaluate guards.

Resource perspective The resource perspective mainly comprises of the work schedules of the resources. The number of resources available to the process is a modification of the initial marking. The task controller computing the resource arousal level and thus the duration of a task is discussed at the control-flow perspective (the computation is in the task controller).

A work schedule is a list of reals denoting the length of the intervals of being present and absent. An integer denotes at which index the schedule currently is, and a timestamp denotes at what time the last change took place. For instance, if we have `(2, [11.3, 4.1, 7, 4], 15.4)`, the "2" denotes that we are at index 2, i.e., the value "7", the values between "[" and "]" are the encoding of the work schedule (available, unavailable), and the "15.4" at the end denotes the timestamp of the last change. Note that since CPN Tools does not allow

guards based on time[5], we cannot directly use the time of the last change added to the duration of the interval as a guard for a transition making a resource (un)available to the process.

Control-flow perspective Within the control-flow perspective, certain parts of the process model do not always contribute to a case. For instance, for an OR construct only the selected branches (at run-time) contribute. To prevent the explicit modelling of all possible paths through the OR, we have introduced the notion of true/false tokens. If the true/false token is true, then that part of the process contributes to the case; otherwise it does not [19].

To be able to cope with multiple cases, we add to each token the notion of a case identifier. Furthermore, this means that transitions for synchronising a subprocess, e.g., the parallel execution of a number of tasks, are now guarded with the requirement that they can only synchronise if all the tokens have the same identifier. With the aforementioned directed acyclic graph of the Process Tree, we might have tokens with the same identifier but belonging to different instantiations of the subprocess. To solve this issue, we unfold the directed acyclic graph of the Process Tree into a tree prior to applying the transformation.

In order to obtain timing information from the simulation, we have extended every token with a timestamp, duration, and performance information about the sojourn time, processing time, queue time, and wait time. The timestamp denotes when a token entered a subprocess rooted at the node. The duration denotes the processing time for a task and is updated just before the tasks starts. The duration of a task is computed by the controller of a task since the duration is based on the task and the arousal level of the allocated resource. Finally, the performance information is used to obtain information about subprocesses and can be used to compute the performance characteristics of a node based on its children. The sojourn time is the time from the start of a subprocess for a case until the end of that subprocess, processing time is the total amount of time resources spent working on a case within a subprocess, queue time is the total amount of time a case waited for a resource in a subprocess, and wait time is the time spend in synchronising parallel branches for a subprocess.

Nodes All nodes offer the same interface places to their parent in the CPN encoding. The interface places offered to the parent are: *scheduled*, *started*, *withdrawn*, and *completed*. Scheduled denotes that a subprocess may start. Started denotes that a subprocess actually has started. Withdrawn is specifically for accommodating events, in which case all events are scheduled at the same time and the event that starts first withdraws the others. Finally, completed denotes that the subprocess has finished its execution. The interface places are encoded using fusion places in CPN Tools [3].

Connected to the interface places are some standard transitions. The *withdraw* transition fires when a token is received on interface place withdrawn, and

[5] Guards are only reevaluated when tokens are added/removed in the surrounding places.

we have not started the execution of this subprocess. If the subprocess has already been started, then the withdrawn token is forwarded with high priority to the children. By using priorities on the transitions, we guarantee that nothing ordinary can happen between obtaining and forwarding the withdrawn token. When a subprocess is scheduled with a true token, then first the *init* transition is fired. The init transition is linked to another fusion place which forms a hook for a controller to notice that a subprocess has been scheduled. Similar, we have a *final* transition to consume the token after a controller has been notified on the completion of this subprocess. Finally, there is a *skip* transition in case a false token is received in the scheduled place.

Tasks As tasks do not have children, we cannot receive a withdraw token after starting the task, i.e., we can only start the task after all unprocessed withdraw tokens have been processed. In case we receive a true token, the task signals the controller of this task being scheduled. As soon as the task is allowed to start, a token is produced signalling that the initialisation has been finished, and the task starts and signals its parent by producing a token on the started interface place. Afterwards, the task signals the task controller that it needs a resource and a duration for its work item. The task controller adds the work item to the list of currently unallocated work items. As soon as a resource has been allocated to a work item (by the *allocation controller*, which we explain later on), the task controller notifies the task and determines the processing time for the task based on the arousal level of the allocated resource. After the duration has elapsed, a token is made available and the task can complete. During the completion of a task, the used resource is released, the read and written variables are updated, and the statistics of the task are computed.

The statistics for a task are computed as follows: The queue time is the elapsed time between a token entering the task's subprocess and the moment a resource has been allocated to the work item. The processing time is the duration of a task. The wait time for a task in isolation is 0 by definition. Finally, the sojourn time is the sum of the queue time and the processing time, i.e., the total time spent in the task subprocess.

Blocks Blocks have the same set of places exposed to their parent as a task has, e.g., a block is also scheduled, and a block signals her parent that she has completed. Dependent on the type of block, different children receive a true token based on different information, e.g., in case of an AND block all the children receive a true token, in case of a XOR block only the first child for which the expression evaluates to true receives a true token, etc.

Due to space limitations, we show the transformation of the XOR block and the LOOPXOR block. The reason for this is that the AND, OR, and DEF blocks can be easily inferred from the XOR block. In turn, the SEQ and LOOPDEF blocks can be easily inferred from the LOOPXOR block. For the encoding of the EVENT block, we refer the reader to [2].

The most interesting part of the XOR is in the selection of which children should receive a true token and which should receive a false token. The relevant

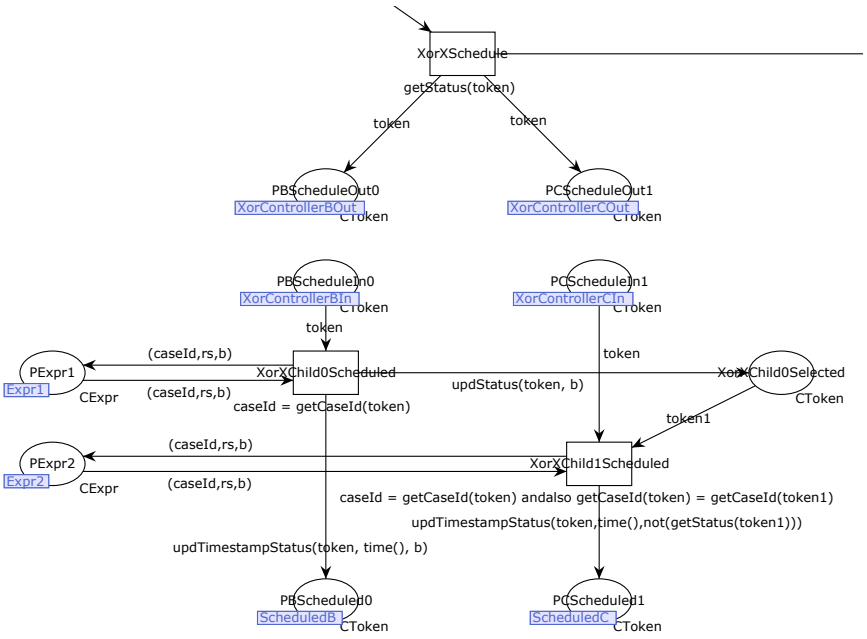

Fig. 8: Fragment in XOR dealing with guards.

part is depicted in Fig. 8. After the XOR has been initialised and scheduled, the XOR signals the fragments corresponding to the guards in the data perspective that it is going to schedule its children. After these fragments are done, the expressions are evaluated in the order of the outgoing edges. If an expression evaluates to true, then we forward a true token to the corresponding child and all other children obtain a false token. We have a special case when all the expressions evaluate to false. In that case, we forward, similar to YAWL [20], a true token to the last child (instead of a false token).

For the statistics for the XOR, we take the statistic of the child which received a true token, i.e., we do not want non-executed subprocesses to interfere with the statistics.

For the LOOPXOR, the most interesting part is the iteration of the loop (Fig. 9). If a loop is scheduled, we first schedule the *do* child of the loop. When the *do* child has completed, we schedule both the *redo* child and the *exit* child. Based on the guard of the *redo* child, we either send the true token to the *redo* child or the *exit* child, the other will be send a false token. Note that we employ a similar strategy as for the XOR, i.e., if all guards are false then the *exit* child receives a true token hence we only need to evaluate the guard of the *redo* child. If the *redo* child has received the true token, we execute the *do* child again. If the *exit* child received the true token, then we can exit the loop.

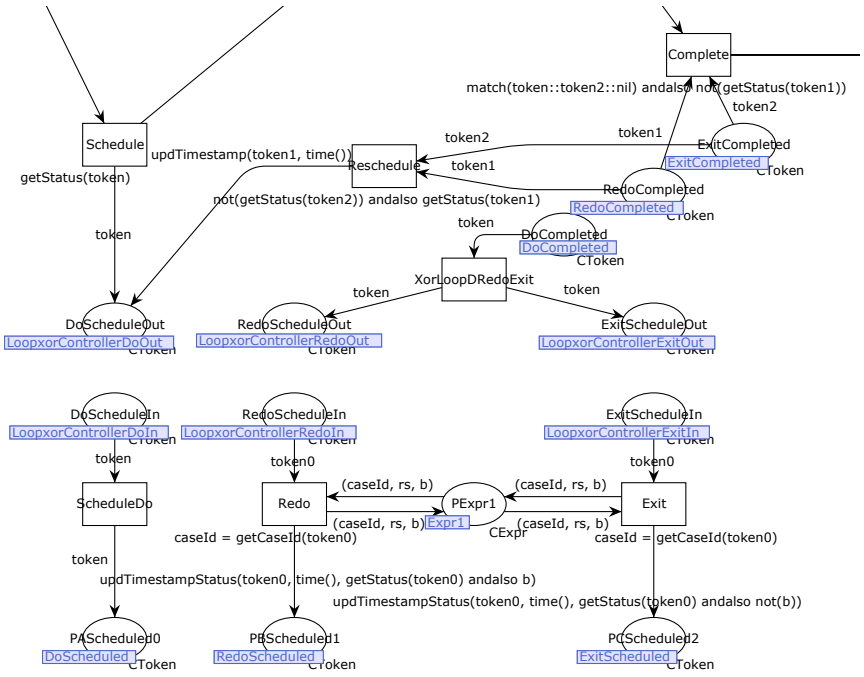

Fig. 9: Fragment in LOOPXOR dealing with guards.

The statistics of the LOOPXOR are stored in the token of the *exit* child. In the tasks, we already increment the values for the different performance measures and due to the fact that the children of the LOOPXOR are sequential, e.g., *do*, *redo*, *do*, and *exit*, there is no need to aggregate the values.

Environment Perspective The environment controller currently consists of the arrival controller only. The arrival controller realises a token generator and a token de-generator. The token generator generates new cases based on the distribution of the arrival process. The token de-generator removes finished cases from the model.

Extra Controllers A controller not belonging exclusively to any of the aforementioned perspectives is the *allocation controller*. The allocation controller allocates work items to resources. It operates on the global list of work items, which contains all cases waiting to be processed by an activity, and on the list of currently available resources to the process. The allocation can either push or pull items by taking either the work item or resource point of view. For instance, if we take the resource point of view (pull), then the resource selects the work

item most desired by her. When taking the work item point of view (push), we select the resource best fitting the work item.

5 Case study

We have taken the building permit process of two Dutch municipalities, M_A and M_B, participating in the CoSeLoG project. Instead of simulating the entire building permit process, we focus in particular on the "Bezwaar en Beroep" (*objection and appeal*) subprocess[6]. This subprocess allows people to object and appeal to the possible conferment[7] of a building permit, and starts after the municipality has published the disposal[8] of a building permit, which gives people the opportunity to object or appeal before the disposal becomes a conferment. The disposal of a building permit means that the municipality agrees with the building permit but it is not yet definitive, i.e., based on objections and appeals the municipality may decide to disagree with the building permit.

In our case study, we want to show *Petra* can indeed analyse a family of process models and find a better process model. From the municipality event logs, we obtained the different perspectives for each of the municipalities. These perspectives have been combined into a Process Tree using the aforementioned configuration options. In order to verify that the perspectives were obtained correctly, we first try to reproduce the behaviour recorded in the original logs from the municipalities. This way we can verify that the perspectives are encoded correctly.

Verifying perspectives The characteristics of the logs are listed in Table 1. The log spans a time period of roughly 2 years. The log consists of *create* events and of *completed* events, and every event has an executor associated with it. The occurrences of the different event classes varies significantly. The least occurring event class occurs just once, while the most occurring event class occurs 262 times for M_A and 451 times for M_B. We only estimate processing times of activities with at least 3 observations in the log, this to have somewhat reliable values for the sojourn time of events whilst not disposing too many activities.

From the event logs, we have constructed Process Trees only consisting of the control-flow perspectives. Since all the choices in the Process Tree are binary, we use variables which with a certain probability are "0" or "1" denoting left and right respectively.

Apart from the flow of the different cases, we also need to estimate the resource behaviour. First, we estimated the work schedule of the resources. The work schedule is estimated based on observations from the log. We have taken the timestamps of all events and made the following assumptions; (1) people work in shifts, (2) there is a morning shift and an afternoon shift, (3) as there

[6] The case study data can be found at: https://svn.win.tue.nl/trac/prom/browser/Documentation/Petra

[7] Conferment means that a building permit is granted to the requester.

[8] Disposal means that a building permit is about to be granted to the requester.

Table 1: The characteristics of the logs used in the case study

Characteristics	Cases	Events	Event classes	Resources
M_A	302	586	15	5
M_B	481	845	23	4

is no clear signal of breaks, we assume the shifts are consecutive, (4) if we have at least one measurement in a shift, we assume that person is available for the process during that whole shift, and finally, (5) we assume a weekly iteration, e.g., if someone worked on a Monday, we assume she can work every Monday.

For the throughput times of the activities, we have used two sources of information. First, by using alignments [21], we aligned the log to the earlier obtained control-flow perspective in order to see where time is spent. Second, there is legislation specifying legal time intervals, e.g., there is a legal limit within which the municipality has to respond. Unfortunately, the law allows for exceptions which are not observable in the log. To take the legal constraints into account, we have explicitly added activities to model these time intervals, and where the law is unclear, we have estimated the time interval. We have also analysed the resource performance between resources when they are allowed to executed the same activity. If there was a significant difference, we have encoded the sojourn time per resource, else we estimated the same sojourn time for all the resources. Furthermore, we have abstracted from outliers.

Using the aforementioned information, we were able to construct the simulation models for both M_A and M_B. In the simulation, we used push allocation, FiFo queues for the work items, a warmup period and replication length of $150,000$ steps in the simulator, and, we generated 30 replications for each.

Prior to comparing the simulation results to the logs, we first have to compute the statistics of the log. Since the log can be seen as a single long replication, we used batch means to be able to compute replications and to be able to compare the log to the simulation results. See [22] for an overview of the batch means method. Fig. 10 shows the results, where M_A is shown left and M_B is shown right and times on the y-axis are in hours.

As one can clearly see, there is overlap in the box plots of the logs and the simulations. Hence, we cannot conclude that our simulation model and reality differ in an unacceptable manner.

Finding a better model After illustrating that the models used in *Petra* can indeed mimic reality, we now combine the models from M_A and M_B into a single Process Tree. This Process Tree is the union of M_A and M_B but allows for more than just M_A and M_B. We want to use this combined Process Tree to improve the sojourn time for M_A by letting *Petra* automatically traverse the family of process models.

Not all combinations make sense, e.g., M_A will not hire the employees of M_B. Hence, we limit the configurability of the Process Tree. This means that we take the characteristics of the employees of M_B into account, e.g., work speed,

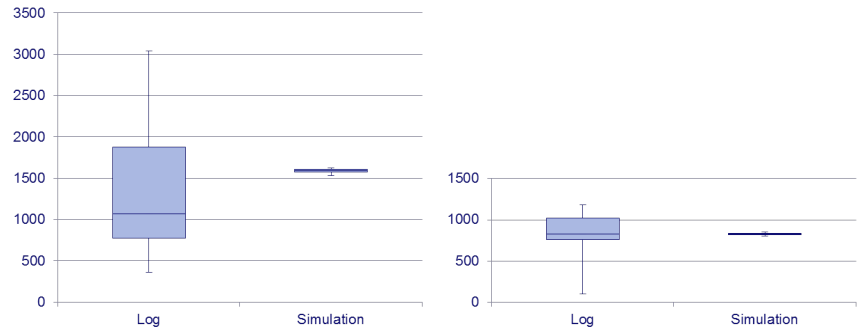

Fig. 10: Validation results.

Table 2: Case study results.

	1	2	3	4	5	6	7	8
540	A	A	A	B	A	B	B	B
630	A	A	B	A	B	A	B	B
770	B	A	A	B	B	A	B	A
Avg	1488.78	1572.48	1570.91	806.60	1496.11	1217.32	803.67	1224.50
Std. dev.	29.55	33.82	18.58	22.93	24.81	34.80	20.24	33.10

but do not put the employees of M_B into the Process Tree. Furthermore, since the building permit process is heavily subjective to legal requirements, there is not much room to change what is done but there is room to change how it is done. Therefore, we focus on the activities *540 Objection to disposal submitted*, *630 Appeal set*, and *770 Establish decision phase original decree*, since these activities embody part of the significant difference between M_A and M_B, and are related to how things are done.

With the focus on the activities *540*, *630*, and *770*, we have a solution space of 8 models. After *Petra* traversed through the family of process models, transformed these models to CPN models, simulated each of them, and enriched the family of process models, we obtain the results (throughput times of the entire process in hours) as in Table 2.

As one can see, working on the activities *540* and *770* in the same way as M_B can already significantly decrease the sojourn time for M_A. Furthermore, one can clearly see that changing *770* and *630* without changing *540* will not yield any significant improvements. Finally, *630* does not have any significant impact on the performance of the process.

6 Conclusion and future work

Using *Petra*, we can take a family of process models, as captured by a Process Tree, traverse this family, and enrich every process model from this family with

KPIs using external analysis tool like CPN Tools. Based on these KPI values, a process owner can then decide which process model suits her best.

Petra is generic and extensible. The genericity is achieved by not limiting our set of performance indicators and set of properties. The extensibility is achieved by allowing any tool to be used. Currently, we have two tools in our framework: CPN Tools and L-SIM. Finally, the implementation of our framework has been applied in a case study using the data from two municipalities resulting in an improved model. The case study shows that it is indeed possible to obtain an improved model. A technical report supporting this paper can be found at [2].

Our framework can be extended in various of directions. Currently, the simulation of each of the CPN models is a time-consuming task. Although *Petra* is multi-threaded, in the experiments, simulating a single CPN model took around 3 days on a core of 2.80 GHz. Since simulation takes such a long time, we would like to minimise the amount of Process Trees to be analysed. In our current implementation, we naively iterate through all possible instantiations of the configurable process model. This means that equivalent models, obtained by different configurations, are analysed multiple times. Defining equivalence classes on configurations and taking these into account in the iteration through the Process Trees could already result in a significant speed up. Another approach to minimising the amount of to-be-analysed models, is to have knowledge beforehand on the performance measures a user wants to optimise. To exploit this knowledge, we also want to have different (faster but imprecise) analysis tools in our framework. This way it would be possible to obtain quick estimates in the relevant measure to decide whether simulating the Process Tree will be worthwhile. Also, simulations could be aborted once it is clear that the model under investigation will never be Pareto optimal. With the use of *Petra*, these things can be easily incorporated without changing any of the aforementioned.

References

1. Schunselaar, D.M.M., Verbeek, H.M.W., Aalst, W.M.P. van der, Reijers, H.A.: Creating Sound and Reversible Configurable Process Models Using CoSeNets. In Abramowicz, W., Kriksciuniene, D., Sakalauskas, V., eds.: BIS. Volume 117 of Lecture Notes in Business Information Processing., Springer (2012) 24–35
2. Schunselaar, D.M.M., Verbeek, H.M.W., Aalst, W.M.P. van der, Reijers, H.A.: Petra: Process model based Extensible Toolset for Redesign and Analysis. Technical Report BPM Center Report BPM-14-01, BPMcenter.org (2014)
3. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009)
4. Westergaard, M.: Access/CPN 2.0: A High-Level Interface to Coloured Petri Net Models. In Kristensen, L.M., Petrucci, L., eds.: Petri Nets. Volume 6709 of Lecture Notes in Computer Science., Springer (2011) 328–337
5. Gottschalk, F., Aalst, W. M. P. van der, Jansen-Vullers, M.H., Verbeek, H.M.W.: Protos2CPN: Using Colored Petri Nets for Configuring and Testing Business Processes. International Journal on Software Tools for Technology Transfer **10**(1) (2008) 95–110

6. Aalst, W.M.P. van der, Nakatumba, J., Rozinat, A., Russell, N.: Business Process Simulation. In Brocke, J., Rosemann, M., eds.: Handbook on Business Process Management 1. International Handbooks on Information Systems. Springer Berlin Heidelberg (2010) 313–338
7. Netjes, M., Mansar, S.L., Reijers, H.A., Aalst, W.M.P. van der: Performing Business Process Redesign with Best Practices: An Evolutionary Approach. In Filipe, J., Cordeiro, J., Cardoso, J., eds.: ICEIS (Selected Papers). Volume 12 of Lecture Notes in Business Information Processing., Springer (2007) 199–211
8. Netjes, M., Reijers, H.A., Aalst, W.M.P. van der: The PrICE Tool Kit: Tool Support for Process Improvement. (2010)
9. Essam, M.M., Mansar, S.L.: Towards a Software Framework for Automatic Business Process Redesign. ACEEE International Journal on Communication **2**(1) (March 2011) 6
10. La Rosa, M., Lux, J., Seidel, S., Dumas, M., Hofstede, A.H.M. ter: Questionnaire-driven Configuration of Reference Process Models. Advanced Information Systems Engineering **4495** (2007) 424–438
11. Hallerbach, A., Bauer, T., Reichert, M.: Capturing Variability in Business Process Models: The Provop Approach. Journal of Software Maintenance and Evolution: Research and Practice **22**(6-7) (November 2010) 519–546
12. Rosemann, M., Aalst, W.M.P. van der: A Configurable Reference Modelling Language. Information Systems **32**(1) (2007) 1–23
13. La Rosa, M., Dumas, M., Hofstede, A.H.M. ter, Mendling, J.: Configurable multi-perspective business process models. Inf. Syst. **36**(2) (2011) 313–340
14. Gottschalk, F.: Configurable Process Models. PhD thesis, Eindhoven University of Technology, The Netherlands (2009)
15. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. In Dumas, M., Reichert, M., Shan, M.C., eds.: BPM. Volume 5240 of Lecture Notes in Computer Science., Springer (2008) 100–115
16. Gagne, D., Shapiro, R.: BPSim 1.0. http://bpsim.org/specifications/1.0/WFMC-BPSWG-2012-01.pdf (Feb 2013)
17. Aalst, W. M. P. van der, Hee, K. M. van: Workflow Management: Models, Methods, and Systems. The MIT Press (January 2002)
18. Yerkes, R.M., Dodson, J.D.: The Relation of Strength of Stimulus to Rapidity of Habit-Formation. Journal of Comparative Neurology and Psychology **18**(5) (1908) 459–482
19. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR (September 1999)
20. Hofstede, A.H.M. ter, Aalst, W.M.P. van der, Adams, M., Russell, N., eds.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2010)
21. Aalst, W.M.P. van der, Adriansyah, A., Dongen, B.F. van : Replaying History on Process Models for Conformance Checking and Performance Analysis. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery **2**(2) (2012) 182–192
22. Fishman, G.S.: Grouping Observations in Digital Simulation. Management Science **24**(5) (1978) pp. 510–521