An analytical method for assessing business processes

Citation for published version (APA):

Document status and date:
Published: 01/01/1998

Publisher Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 04. Sep. 2020
An Analytical Method for Assessing Business Processes

by

K.M. van Hee and H.A. Reijers

ISSN 0926-4515

All rights reserved
editors: prof.dr. R.C. Backhouse
        prof.dr. J.C.M. Baeten

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Reports 98/09
Eindhoven, June 1998
AN ANALYTICAL METHOD FOR ASSESSING BUSINESS PROCESSES

K.M. van Hee, H.A. Reijers
Department of Mathematics and Computing Science, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, telephone: +31 40 2472733,
e-mail: {wsinhee, hreijers}@win.tue.nl

Abstract
Often, the redesign of business processes (BPR) takes place on intuitive grounds. In this paper, an
alternative approach is propagated. To determine the effectiveness of a redesign measure the model of
the business process in question can be analysed. From the many performance indicators that drive
redesign efforts we concentrate on the throughput time of a business process. A graphical grammar is
introduced to construct complex business process models. The Petri net formalism is applied to
capture the structure of the process, while probability distributions are used to reflect the behaviour of
the individual tasks within the process. As it is our aim to incorporate techniques as the one discussed
in a tool that can support the redesign of business process, emphasis is put on the efficiency of the
presented algorithms.

Keywords: Business Process Modelling; analysis of workflows; Petri nets; throughput time.

1. Introduction

How can we improve our business process? Today, this question is on the mind of
many people involved in business operations. From an economic perspective it is
simple to see why asking this question is useful. When an organisation is able to
deliver services or produce goods at a lower cost, at a higher speed, or of better
quality it can be more competitive. And therefore, more profitable.

Examples of organisations that have successfully improved their processes are well
known. Over the last ten years, both management and scientific journals have
reported frequently on organisations that have adopted (radically) new ways of
doing business. In an attempt to distinguish common characteristics of successful
redesign efforts, scholars and redesign practitioners alike have issued a considerable
amount of methodologies and theoretical frameworks ([10, 13]).
Obviously, the real challenge of a redesign framework is in the development of a prescriptive notion, setting forth how to actually improve a business process. Those who examine existing material in search of such a notion may experience feelings of disappointment. Redesign directions are commonly formulated in very general terms. Even worse, the justification of suggested measures relies heavily on the fact that a similar approach has worked before in a somewhat similar situation. What is often missing is an accurate, quantitative estimate of the effect of the redesign step. One of the few promising steps in this direction is described by Buzacott [8].

The aim of this paper is to make a contribution to a more rational, a priori evaluation of redesign measures. We do so by putting the focus on the analysis of a business process model. Prior to the actual effectuation of a redesign measure, its benefits can be assessed by analysing the model that reflects the redesigned business process. An analysis technique that can be used to generate a rapid estimation of such a criterion is the main subject of this paper.

Our approach is limited in at least two ways. In the first place, only the structure of a business process is taken under consideration. A business process is seen as a complex of more or less autonomous tasks. Between these tasks precedence relations exist. For example, the task of granting a loan to a client should preferably take place after the credentials of the potential conman have been checked reassuringly. A structural view on a business process merely involves this task topology. It does not take into account, for example, the quality of the involved resources.

The second limitation is in our choice of evaluation criteria. Several performance indicators can be distinguished that drive redesign efforts. We pick one important of the many alternatives: the throughput time of the business process. This is the total amount of time between the moment that a job is started and the moment it is completed. The throughput time of a job is the sum of its service time and its waiting time. Waiting time for a job is created when no task can be executed due to the unavailability of resources. When there is at least one task processing a job, service time is built up. In this paper we will assume that there is an infinite amount of resources available for the execution of each job. The Under this assumption, the throughput time is equal to the service time. Very often, a low or stable throughput time is a desirable or even necessary characteristic of a business process. In some countries, for instance, national regulations are in violation when the governmental review of a tax form exceeds a year. Note that the only known technique that can be
used to determine throughput time in case of finite resources is *simulation*. Several tools are available that enable the simulation of business processes (e.g. ExSpect [7]).

The formalism that we use in this paper to model business processes is based on Petri nets. More precise, the business processes we consider can be modelled as *workflow nets* ([2]). Petri nets allow for powerful modelling constructions. Often before, Petri nets have been applied to model business processes ([1], [11], [16]). They offer the expressive power of the PERT technique ([20]), but have the additional capability of choice constructions. Similarly, Petri nets advance beyond Markov chains as they offer parallelism as well. The way in which we deal with time in our models of business processes is in line with the Timed Petri Nets tradition ([3], [6], [12], [14], [21], [23], [24], [28]). However, we do not restrict the modelling of service time to fixed time durations ([23], [24], [28]) or time intervals ([3], [21]). For the sake of realism, we choose to model service time with probability distributions that can be freely chosen. In other words, we are not restricted to a specific type of probability distribution. In traditional stochastic Petri Nets, exponential distributions would have to be used for modelling service time ([4], [5], [12], [22]), possibly extended with options for timeless processing ([6]). More recent stochastic models allow for combinations of discrete, geometric and uniform distributions ([27]). Resembling the restrictions in stochastic Petri Nets, modelling time in PERT graphs is constrained to the application of normal, gamma, or beta distributions ([19], [26]). We will use probability theory and the Fast Fourier transform to tackle the computational burden that arises from keeping the service time characteristic free.

The paper is organised as follows. First, we introduce some definitions and concepts that we will use to model and analyse processes. We will present four composition primitives to construct the model of a business process. We will also indicate how to model the dynamic behaviour of such a net. After that, we will show how the calculation of the throughput time takes place for the application of each of these composition primitives. Special attention will be given to the efficiency of these computations. Next, we will show how these results can be applied on *workflow* processes. Finally, we present our conclusions and directions for further research.
2. Modelling a business process

2.1 Petri Nets

To model business processes we will use a framework based on high-level Petri nets ([15]). The classical Petri net is a directed bipartite graph with two node types called places and transitions. Places are graphically represented by circles and transitions by rectangles. As shown by Van der Aalst and Van Hee [1] Petri nets allow for a natural representation of business processes. Process tasks are modelled as transitions and causal dependencies between tasks as places.

Definition (Petri Net) A Petri net is a triple \((P, T, F)\):
- \(P\) is a finite set of places,
- \(T\) is a finite set of transitions \((P \cap T = \emptyset)\),
- \(F \subseteq (P \times T) \cup (T \times P)\) is a set of arcs (flow relation)

A place \(p\) is called an input place of a transition \(t\) if and only if there exists a directed arc from \(p\) to \(t\). Place \(p\) is called an output place of transition \(t\) if and only if there exists a directed arc from \(t\) to \(p\). We use \(\bullet t\) to denote the set of input places for a transition \(t\). The notations \(t\bullet, \bullet p\) and \(p\bullet\) have analogue semantics, for example \(p\bullet\) is the set of transitions sharing \(p\) as an input place. At any time a place contains zero or more tokens, graphically depicted as black dots. The state, often referred to as marking, is the distribution of tokens over places. The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net. They change the state of the net according to the following firing rule:

(i) A transition \(t\) is said to be enabled iff each input place \(p\) of \(t\) contains at least one token.

(ii) An enabled transition may fire. If transition \(t\) fires, then \(t\) consumes one token from each input place \(p\) of \(t\) and produces one token for each output place \(q\) of \(t\).

With a Petri net we are able to model a wide variety of processes. All kinds of logistic concepts applied in actual business like sequential processing, triage, and parallelism can be modelled. On the other hand, there is expressive room for constructions that we do not need. To reflect somewhat more accurately the nature of a business process we restrict ourselves to a smaller class of Petri nets, the so-called workflow nets ([2]).
Definition (Path) Let PN be a Petri Net. A path C from a node \( n_1 \) to a node \( n_k \) is a sequence \( <n_1, n_2, \ldots, n_k> \) such that \( <n_i, n_{i+1}> \in F \) for \( 1 \leq i \leq k - 1 \).

Definition (Strongly connected) A Petri net is strongly connected iff, for every pair of nodes (i.e. places and transitions) \( x \) and \( y \), there is a path leading from \( x \) to \( y \).

Definition (Extended net, source place, sink place) When a Petri net \( PN = (P, T, F) \) has exactly one place \( i \) such that \( \bullet i = \emptyset \) and exactly one place \( o \) such that \( o \bullet = \emptyset \), we define the extended net \( PN^* = (P, T \cup \{t^*\}, F \cup \{<o, t^*>, <t^*, i>\}) \) where \( t^* \) is an added transition, \( t^* \notin T \). We call \( i \) the source place and \( o \) the sink place of \( PN \).

Definition (Workflow net) A Petri net \( PN = (P, T, F) \) is a workflow net iff:

(i) \( PN \) has exactly one source place and exactly one sink place.
(ii) \( PN^* \) is strongly connected.

The first requirement in the definition of a workflow net reflects the natural begin and termination of a business process. A business process is often initiated by an event, like the service request of a customer or a parts order from another department. That event is to be handled in such a way that a desired end situation is reached, like the completion of a service or the fulfilment of an order. A ‘flow of work’ can be distinguished between these two points, hence the name workflow. The second requirement in the definition ensures that there are no dangling workflow tasks or causal dependencies. In other words, transitions or places are not permitted to be part of a business process model if they do not contribute to the processing.

2.2 Constructing a business process

At this point, we are able to start building business processes models. We will do so by taking an iterative approach, starting with a very primitive net and refining it locally with successive construction steps. This is very similar to constructing a graph grammar ([18]). Another way to look at it, is that each new construction step adds another hierarchical level in the Petri net at hand ([15]). Keeping in mind the construction tree of the business process model, later on we will be able to execute an algorithm to compute the throughput time of complex workflow nets. In terms of a graph grammar, this tree is called the derivation.
In Figure 1 a Petri net is depicted that will be used as the starting point of each business process construction. Looking at our construction approach as a graph grammar, this net would be the initial non-terminal ([18]). The net consists of one transition, a source place, a sink place, and relations between them. It is easy to see that this Petri net is a workflow net. We will refer to this specific Petri net as SN, for start net.

We assume that the names of all places and transitions in the nets to be analysed are unique, except for $i$ and $o$. Furthermore, four building blocks are available for the construction of a more complex net. These building blocks are depicted in Figure 2. The choice for these building blocks is based on their recognition by the Workflow Management Coalition as the main routing constructs in workflow processes ([17]).
The construction rule that should be followed to construct a more complex net 
$C = \{P_c, T_c, F_c\}$ from a simpler net $S = \{P_s, T_s, F_s\}$ is as follows:

(i) Identify within net $S$ a subnet with transition $t$, input place $p$ and output
place $q$ that is isomorphic with the start net, i.e.
\[ t = \{p\}, \]
\[ t* = \{q\}, \]
Select one of the building blocks $B = \{P_b, T_b, F_b\}$ with source place $i$ and sink
place $o$;

(ii) Replace the subnet by the building block which results in the new complex
net $C$, i.e.
\[ P_c = P_s \cup (P_b \setminus \{i, o\}) \]
\[ T_c = T_s \cup T_b \setminus \{t\} \]
\[ F_c = (F_s \setminus \{<p, t>, <t, q>\}) \cup ((F_b \setminus \{<i, v> | v \in T_b\}) \setminus \{<v, o> | v \in T_b\}) \cup \{<p, v> | <i, v> \in F_b\} \cup \{<v, q> | <v, o> \in F_b\} \]

We shall clarify both the replacement rule and the introduced building blocks
informally. (Note that this construction rule can be seen as the single production rule
in a context-free graph grammar.) Applying the first building block, the sequence, on
the start net would result in a net that has two sequential tasks. In other words, the
insertion of the sequence building block extends the original subnet with a sequential
task. The sequencing concept is a very common one in business processes. Recall the
example in the introduction where the credentials of a client have to be checked
before a loan is granted.

The resulting net allows for yet another application of the construction rule with the
sequence building block. After all, within the resulting net two instances of the
primitive net can be distinguished. (As a consequence, the graph grammar is non-
linear. [18]) The described situation is depicted in Figure 3.
Each of the building blocks can be used on either of the identified instances of the start net. Applying the second building block converts one of the start net instances into a construction where a choice is to be made between the execution of two tasks. This logistic concept is appropriate when a different line of action can be distinguished, depending on the characteristics of the job at hand or other parameters. For instance, financial transactions that exceed a certain amount of money may need to undergo a stricter validation than other ones do. Should the third building block be applied on one of the start net instances, parallel paths are created. This concept is appropriate when tasks in a business process can be executed in any order or even simultaneously. An example would be producing a bicycle frame simultaneously with producing its wheels. The fourth building block, the iteration, is there to create a task that can possibly be repeated. A typical example would be that reworking is required when quality standards are not met.

In Figure 4 the creation of a business process is depicted, as a result of successive applications of building blocks. The initial start net is extended by applying the construction rule with the sequential building block. Next, the upper half of the resulting net is transformed into a choice construction. On the lower half, the parallel composition is applied. Finally, the right path of the parallel construction is modified with the iteration building block.
Note that we have, by now, informally introduced a graph grammar. Given a Petri net $PN_1 = (P, T, F)$ we have the following additional notations:

- $PN_1 \rightarrow PN_2$: there is an application of the construction rule on $PN_1$ using one of the four building blocks such that $PN_2$ is constructed
- $PN_1 \rightarrow^* PN_2$: there is a sequence of applications of the construction rule on $PN_1$, each time using one of the four building blocks, such that $PN_2$ can be constructed

**Theorem (WF-net preservation)** If $BP = (P, T, F)$ is the model of a business process such that $SN \rightarrow BP$, then $BP$ is a workflow net.
Proof.
Suppose we have workflow net $WF$ and building block $B$ is inserted in it by applying the construction rule on a start net instance with source place $a$ and sink place $b$. The resulting net is a Petri Net $PN = (P, T, F)$.

a) $WF$ has exactly one source and one sink place and so has each of the building blocks. The construction rule prohibits the destruction of the original source and sink places of $WF$ and does not introduce new source or sink places. As a consequence, $PN$ has exactly one source and exactly one sink place.

b) Consider an arbitrary pair of places $p$ and $q$ within $PN^*$. We can distinguish four different situations for which we will prove that there is path from $p$ to $q$ within $PN^*$ (strongly connected).

1. $p$ and $q$ are part of $B$: in this case, there may have been a path already from $p$ to $q$ within $B$, which then still exists in $PN$. If not, now there is a path between them that at some point crosses $b$ and then continues its way through $WF^*$ to $a$. The latter holds because there was already such a path in $WF^*$.

2. $p$ and $q$ are part of $W$: the original path between $p$ and $q$ in $WF^*$ is still intact within $PN^*$ if it did not cross $a$ and $b$. Otherwise, the original route can be updated between $a$ and $b$ with a path in $B$ that leads from its source to its sink place.

3. $p$ is part of $B$ and $q$ is part of $WF$. Within $B$ there is a path leading from $p$ to the sink place of $B$. In $WF^*$, there is a path from $b$ to any other place within $WF^*$. Consequently, there is a path in $PN^*$ from $p$ to $q$.

4. $p$ is part of $WF$ and $q$ is part of $B$. Within $WF^*$ there is a path leading from $p$ to $a$. In $B$ there is a path from its source place to $q$. Consequently, there is a path in $PN^*$ from $p$ to $q$.

Therefore, on basis of the definition of a workflow net, $PN$ is a workflow net. We have proven that applying the construction rule on a workflow net yields another workflow net. As $SN$ is a workflow net itself, our proof is complete.

\[ \square \]

2.3 Modelling throughput time

With the notions presented in the former section we can model the outline of a business process. The dynamic behaviour of tasks is something we still have to address. Commonly, each time a task is executed for another job its throughput time is different. Causes for the variances in throughput time are manifold. One job may require more work than the other does because it has certain special characteristics.
Also, the varying precision of used machinery may cause differences in throughput time.

In our model of a business process, the throughput time of a task is a discrete random variable. In that way, we abstract from the actual factors that determine the throughput time for a special job instance. We focus, instead, on the distribution characteristic of the task over a longer period of time. In addition, we assume that the throughput time of a specific task is not influenced by the execution of another task. Although time is not a discrete entity, we can always find some discrete scale that fits our need for an accurate representation of time. Obviously, to model the throughput dynamic of tasks their actual throughput behaviour should be determined, for example, on basis of historic performance records of the business process.

**Definition (throughput time, throughput time probability density)** The throughput time of a task \( t \in T \) within a workflow net \( W = (P, T, F) \) is a discrete random variable \( f_t \). It expresses the distribution over a non-negative time interval from the moment that a task is enabled until the moment it fires. Its matching probability density is denoted with \( f_t \).

The throughput time of a workflow net \( W = (P, T, F) \) is a discrete random variable \( W \). It expresses the distribution over a non-negative time interval from the moment that a token is available in the source place of \( W \) until it has arrived in the sink place of \( W \). Its matching probability density is denoted with \( f_W \).

For each probability density \( f \) holds:
- \( f : \mathbb{N} \rightarrow [0,1] \),
- \( \sum_{j=0}^{\infty} f(j) = 1 \).

The throughput time probability density in a specific point \( x \) can be interpreted as the probability that execution of task \( t \) lasts \( x \) time units. For what follows, instead of using the full phrase ‘throughput time probability density’ we will use the shorthand *throughput density*. Closely related to the throughput density of a task or workflow is its *distribution function.*
Definition (throughput time distribution) Given the throughput time $t$ of a task $t \in T$ within a workflow net $W = (P, T, F)$, its throughput distribution $F_t$ is defined as follows:

$$F_t(z) = P(t \leq z) = \sum_{j=0}^{z} f_t(j), \text{ for } z \in \mathbb{N}.$$ 

As we have stated before, the rest of this paper will show how the workflow net of a business process enriched with a throughput notion can be used to compute the throughput time of the entire business process. In the following section, each building block will be discussed separately.

3. Computing throughput time

3.1 Sequence building block

Based on the throughput densities of each of the tasks within a building block, we start to show that we can compute the throughput density for each of the building blocks. We start with the sequence building block.

Consider the sequence building block $B$ in Figure 5 with two tasks $s$ and $t$. We want to compute $f_B$, given $f_s$ and $f_t$.

Let $y \in \mathbb{N},$

$$f_B(y) = \{ \text{structure workflow net; no waiting time} \}$$

$$P(s + t = y)$$

$$= \{ \text{all possible throughput distributions over tasks } s \text{ and } t; y \text{ non-negative } \}$$
\[
\sum_{i=0}^{y} P(s = i \land t = y - i)
\]

= \{ s and t stochastically independent \}

\[
\sum_{i=0}^{y} P(s = i) \cdot P(t = y - i)
\]

= \{ definition convolution: \( f \otimes g (x) = \sum_{j=0}^{x} f(j) \cdot g(x - j) \) \}

\[
f_s \otimes f_t(y)
\]

If either the domain of \( f_s \) or \( f_t \) is countably infinite then a straightforward computation of the function \( f_s \otimes f_t \) will take an infinite amount of time. Recalling the practical nature of the throughput time we can feel confident about assuming that both \( f_s \) and \( f_t \) are finite. As a result, there will be an upper bound for the execution times of each of the respective tasks \( s \) and \( t \). In other words, each task time has a fixed upper bound.

Definition (Upper bound execution time)
The upper bound \( u_t \in \mathbb{N} \) of task \( t \) within a workflow net is the smallest value such that for all \( j \in \mathbb{N} \) and \( j \geq u_t \) holds that \( f_t(j) = 0 \).

A straightforward computation of the function \( f_s \otimes f_t \) would require at least \( u_t \cdot u_s \) multiplications. (Note that the upper bound of \( f_s \otimes f_t \) is \( u_t + u_s - 1 \).) Consequently, the computation of the throughput density of a sequential building block can easily run out of hand. To constrain the computation effort drastically we propose to use a more efficient computation method for the convolution: the Discrete Fourier Transform ([9]).

Definition (Discrete Fourier Transform)
The Discrete Fourier Transform (DFT) of the vector \( \tilde{a} = (a_0, a_1, ..., a_{n-1}) \) is given by the vector \( \tilde{y} = (y_0, y_1, ..., y_{n-1}) \) with

\[
y_k = \sum_{j=0}^{n-1} a_j e^{(2\pi i/n)kj}, \text{ for } 0 \leq k < n.
\]

When the vector \( \tilde{y} \) is the Discrete Fourier Transform of vector \( \tilde{a} \) with length \( n \) we also write \( \tilde{y} = \text{DFT}_n(\tilde{a}) \). Without supporting proof we present the following theorem ([9]):
Theorem (Convolution theorem)

For any two vectors $\vec{a}$ and $\vec{b}$ of length $n$, where $n$ is a power of 2,

$$\vec{a} \otimes \vec{b} = \text{DFT}_{2n}^{-1}(\text{DFT}_{n}(\vec{a}) \cdot \text{DFT}_{n}(\vec{b})),$$

where the vectors $a$ and $b$ are padded with 0's to length $2n$ and $\cdot$ denotes the componentwise product of two vectors.

A pleasant property of both the $\text{DFT}_n$ and its inverse $\text{DFT}_n^{-1}$ is that they can be performed in $\theta(n \log n)$ time by the Fast Fourier Transform. A suitable vector representation for a throughput density is easy to find, by taking the throughput density in $i$ as the $i$th coefficient of the vector. As we know that each throughput density is bounded, the proper minimal length of a representing vector is equal to this upper bound. Applying the convolution on two vector representations of throughput densities will result in a vector representation of the convolution of these throughput densities. Therefore, we can compute a vector representation of $f_s \otimes f_t$ in $\theta(n \log n)$ time, with $n$ the smallest power of two that is at least twice as large as the maximum of the upper bounds of tasks $s$ and $t$.

3.2 Iteration building block

In this section the computation of the throughput time of the most complex building block will be presented: the iteration building block. Consider building block $B$ in Figure 6 that consists of tasks $s$, $t$, $u$, and $v$. The choice for either task $u$ or task $v$ after completion of task $t$ is a matter of chance. With $\alpha$ unequal to zero, there is a probability that task $t$ is executed more than once before the completion of $B$. Similarly, it would be possible that task $u$ is executed more than zero times. We assume that the duration of each visit to one of the above tasks is determined by a new, independent draw from the throughput density that belongs to this task. Likewise, draws from the probability function that determine whether task $u$ or $v$ are executed after completion of task $t$ are independent. As a final note, just like we have done in the case of the throughput times, we assume that in practical situations values such as $\alpha$ can be determined.
To simplify the computation of the overall throughput time we will cut up the building block in three smaller parts. These parts, a, b, and c, are presented in Figure 7.

When the output place of part (a) is mapped onto the input place of part (b), and the output place of part (b) is mapped onto the input place of part (c), we have composed the original building block B. We introduce random variables $a$, $b$, and $c$ for the throughput times of the respective parts. Based on the result in the former section, we can establish that the throughput density of building block B is equal to $f_a \otimes f_b \otimes f_c$. The real issue, of course, is the computation of $f_b$. Let us examine this function in more detail.
Let $y \in \mathbb{N}$, 
\[
    f_b(y) = \{ t \text{ will always be executed one time more than } u; \text{ for the sake of clarity, define } \mathcal{Z}_n \text{ as a copy of random variable } \mathcal{Z} \text{ from which the } n^{th} \text{ successive draw from } \mathcal{Z} \text{ takes place; define random variable } \eta \text{ as the number of times that task } u \text{ is executed } \}
\]
\[
    \sum_{n=0}^{\infty} \mathbb{P}(\sum_{j=1}^{n+1} \mathcal{T}_j + \sum_{j=1}^{n} \mathcal{U}_j = y \land \eta = n)
\]
\[
    = \{ \eta \text{ is derived from random variable } \mathcal{I} \text{ that determines whether task } v \text{ or task } u \text{ is executed after completion of task } t; \mathbb{P}(\mathcal{I} = v) = 1 - \alpha; \mathbb{P}(\mathcal{I} = u) = \alpha; \text{ therefore } \mathbb{P}(\eta = n) = (1 - \alpha)^{n-1} \alpha; \text{ random variables independent } \}
\]
\[
    \sum_{n=0}^{\infty} (1 - \alpha)^{n-1} \alpha^n \mathbb{P}(\mathcal{I} = v) \left( \sum_{j=1}^{n+1} \mathcal{T}_j + \sum_{j=1}^{n} \mathcal{U}_j = y \right)
\]
\[
    = \{ \text{definition throughput density } f; \text{ throughput times are based on same throughput density; definition convolution; introduce notation } \bigotimes_{j=1}^{n} a_j = a_1 \otimes a_2 \ldots \otimes a_n \}
\]
\[
    \sum_{n=0}^{\infty} (1 - \alpha)^{n-1} \alpha^n \left( \bigotimes_{j=1}^{n+1} f_\mathcal{T} \otimes \bigotimes_{j=1}^{n} f_\mathcal{U}(y) \right)
\]

No obvious next step leads from here. Now suppose we could derive the DFT of the vector representation of $f_b$ from the DFT's of $\mathcal{T}$ and $\mathcal{U}$? In that case we would know how to use the inverse of the DFT to actually compute $b$ - the vector representation of $f_b$. Because we do not know the proper length of $b$ we denote the index of the DFT, for now, with $l$.

\[
    \text{DFT}(b)
\]
\[
    = \{ \text{recall derivation of } f_b(y); \text{ use } \cdot \text{ for pointwise vector multiplication and } + \text{ for pointwise addition } \}
\]
\[
    \text{DFT}_l \left( \sum_{n=0}^{\infty} (1 - \alpha)^{n-1} \alpha^n \left( \bigotimes_{j=1}^{n+1} \mathcal{T}_j \otimes \bigotimes_{j=1}^{n} \mathcal{U}_j \right) \right)
\]
\[
    = \{ \text{DFT distributes over multiplication and addition } \}
\]
\[
    \sum_{n=0}^{\infty} (1 - \alpha)^{n-1} \alpha^n \text{DFT}_l \left( \bigotimes_{j=1}^{n+1} \mathcal{T}_j \otimes \bigotimes_{j=1}^{n} \mathcal{U}_j \right)
\]
\[
    = \{ \text{convolution theorem; convolution is associative } \}
\]
\[
    \sum_{n=0}^{\infty} (1 - \alpha)^n \text{DFT}_l^n(\mathcal{T}) \cdot \text{DFT}_l^n(\mathcal{U})
\]
This is a fine way to express the DFT of the vector representation of \( f_b \) in terms of the DFT's of \( i \) and \( u \). There is one snag: we still do not know the proper size \( l \) of the vectors we have to "feed" the DFT. Even if, in analogy with the sequential case, we would take the length equal to the smallest \( n \in \mathbb{N} \) such that it is a power of two and both \( n \geq 2u_t \) and \( n \geq 2u_u \) holds, we are still facing an odd problem. We can not expect \( f_b \) to have an upper bound. After all, \( t \) and \( u \) could be executed infinitely often if \( \alpha \) is non-zero. So by choosing \( n \) as proposed we may end up with a vector representation of \( f_b \) that is too short. That is, there may be interesting values of \( f_b \) that will not be represented.

An infinite execution time, of course, is a theoretical situation. In a real business process measures will be taken to prevent an infinite loop. As our process model does not allow for exception handling we will take another approach. We will try to find a 'relevant' length of \( b \) before the actual computation of this vector. Recalling the definition of an upper bound we would be most pleased to find a value \( v \) such that for some very small \( \epsilon \) holds:

\[
\Pr(b \geq v) \leq \epsilon.
\]

Such a value \( v \) can be used as an estimation of the length \( l \) we were out to find. We can find an approximation for \( v \) on basis of the upper bounds of tasks \( t \) and \( u \) and the factor \( \alpha \):

\[
v \geq \frac{\log \epsilon}{\log \alpha} (u_t + u_u) \Rightarrow \Pr(b \geq v) \leq \epsilon
\]

This derivation for \( v \) is based on the observation that each cycle of executions of \( t \) and \( u \) can add at most \( u_t + u_u \) time to the overall throughput time. Therefore, the tasks \( t \) and \( u \) have to be executed at least \( v/(u_t + u_u) \) times to arrive at an execution time that is at least \( v \). The probability of this to happen is \( \alpha^{v/(u_t + u_u)} \), from which follows the above estimation.

However, this estimation for \( v \) may be inaccurate if the upper bounds are far outside the range of regular throughput time values. Therefore, we are looking for an estimation of \( v \) that incorporates a more exact characterisation of the distribution of
values within \( f_i \) and \( f_a \). For this purpose we will use Chebyshev’s inequality, that we present without proof ([25]). This inequality is often used to find a rough upper bound in mostly theoretical applications.

**Theorem (Chebyshev’s inequality)**

For any random variable \( x \) for which \( E x^2 \) exists:

\[
Pr[|x - E_x| \geq c] \leq \frac{\text{var} \ x}{c^2}
\]

With this inequality in the one hand, and the mean and variance of the throughput density \( f_b \) in the other, it can be determined which probability part of the density falls before or after a hypothetical border. As the throughput time for any task \( t \) is denoted by \( t \) we will denote its mean by \( E_t \) and its variance by \( \text{var} \ t \). Recall that we were looking for a way to express the throughput density of part \( (b) \) with throughput time \( b \).

\[
E_b = \{ \text{structure workflow net} \} \frac{Et + \alpha \cdot (Eu + E_b)}{1 - \alpha}
\]

To compute \( \text{var} \ b \), we have to make a small detour first. We can simplify this computation by computing the variance of the throughput time of workflow net \( H \). This auxiliary net is depicted in Figure 8. Note that it is isomorphic with the choice building block that is presented earlier.

\[
\text{var} \ H = \{ \text{calculus; probability definitions} \}
\]
\[ \alpha \text{var} f + (1 - \alpha) \text{var} g + \alpha(1 - \alpha)(E_f - E_g)^2 \]

We can model workflow net \( H \) in such a way that throughput time \( H \) exactly matches \( b \). To accomplish that, the following substitutions should be applied to random variables \( f \) and \( g \) in workflow net \( H \):

- \( f \leftarrow t + u + b \);
- \( g \leftarrow t \).

With the newly found computation of \( \text{var} H \) we can compute \( \text{var} b \).

\[
\text{var} b = \{ \text{apply computation of } \text{var} H \text{ and substitution rules} \}
\]

\[ \alpha \text{var}(t + u + b) + (1 - \alpha) \text{var} t + \alpha(1 - \alpha)(E(t + u + b) - Et)^2 \]

\[
= \{ \text{calculus; previous result for } E \text{ and definition variance} \}
\]

\[ \text{var} t + \alpha \text{var} u + \alpha \text{var} b + \alpha(1 - \alpha)\left( E_u + \frac{E_t + \alpha E_u}{1 - \alpha} \right)^2 \]

\[
= \{ \text{calculus} \}
\]

\[ \frac{\text{var} t + \alpha \text{var} u}{1 - \alpha} + \frac{(E_u + E_t)^2}{(1 - \alpha)^2} \]

Finally, with Chebyshev's inequality we can determine a relevant part of vector \( \vec{b} \) by modifying our original requirement (i) with:

(ii) \( \Pr(b \geq v \wedge b \leq -v) \leq \epsilon \).

This is only apparently a stronger requirement than (i), as \( b \) can never be negative and the sum of positive values of \( f_b \) is 1. Application of Chebyshev on equation (ii) yields the following equalities:

\[ v = c + Eb \]

and

\[ \epsilon = \frac{\text{var} b}{c^2}. \]

From these equalities we can derive that:

\[
\sqrt{v} = Eb + \sqrt{\frac{\text{var} b}{\epsilon}}.
\]
When we take $\varepsilon$ near to zero we have a $v$ such that the accumulated probability of throughput times longer than $v$ is small. Of course, we have to take the smallest power of two that is larger than $v$ to properly apply the DFT.

Concluding, given $f_i$ and $f_u$, we can compute a vector representation of $f_b$ by using the DFT:

$$DFT_v(b) = \frac{(1 - \alpha) \cdot DFT_v(i)}{1 - \alpha \cdot DFT_v(i) \cdot DFT_v(u)}$$

with

$v$ is the smallest power of two such that $v \geq E_b + \sqrt{\frac{\text{var} b}{\varepsilon}}$, and

$$\text{var} b = \frac{\text{var} t + \alpha \text{var} u}{1 - \alpha} + \alpha \frac{(E_u + E_f)^2}{(1 - \alpha)^2}.$$

With the DFT we can compute a vector representation of $f_b$ in $(v \log v)$ time, with $v$ as specified. To appreciate its efficiency we have to establish the computing time of calculating $f_b$ in a straightforward manner. The complexity of this calculation depends on the maximal number of successive times that tasks $t$ and $u$ can be executed. We know that if both $f_i(0)$ and $f_u(0)$ are equal to zero, at most $v$ executions of these tasks are of interest. Any more executions of tasks $u$ and $t$ would result in throughput times that we do not take into consideration. As a result, a straightforward approach requires the convolution of $v$ times the function $f_i$ and $f_u$. This is an operation requiring $O(n^2)$ time, with $n$ the maximum of upper bounds of tasks $t$ and $u$. A comparison with the $O(v \log v)$ time required by our newly found computation method illustrates the efficiency of the latter.

3.3 Parallel building block

The building block we will consider next is the parallel building block. Consider the parallel building block $B$ in Figure 9 with tasks $k, l, m,$ and $n$. Due to the structure of $B$, tasks $l$ and $m$ can be executed in parallel. That is, there is no precedence constraint between tasks $l$ and $m$. When both tasks $l$ and $m$ are ended, task $n$ can be executed. We want to compute throughput density $f_b$, given $f_k, f_l, f_m,$ and $f_n$. 

- 20 -
Let \( y \in \mathbb{N} \),
\[
f_B(y)
\]
= \{ \text{structure workflow net} \}
\[
\sum_{k+l+n=y} P(k = k \wedge l \max m = l \wedge n = n)
\]
= \{ \text{definition max} \}
\[
\sum_{k+l+n=y} P(k = k \wedge ((l = l \wedge m \leq l) \vee (l < l \wedge m = l)) \wedge n = n)
\]
= \{ \text{independent random variables} \}
\[
\sum_{k+l+n=y} f_k(k) \cdot \left( f_l(l) \cdot \sum_{i=0}^{l} f_m(i) + f_m(l) \cdot \sum_{j=0}^{l} f_l(j) \right) \cdot f_n(n)
\]
= \{ \text{assume an } f_x \text{ such that } f_x(l) \text{ is equivalent to the middle term} \}
\[
\sum_{k+l+n=y} f_k(k) \cdot f_x(l) \cdot f_n(n)
\]
= \{ \text{definition convolution} \}
\[
f_k \otimes f_x \otimes f_n(y)
\]

Just as in the case of the iteration building block, we can compute the throughput density of the building block by applying the convolution on three throughput densities. Of course, we still have to unfold the calculation of \( f_x \).
Let $y \in \mathbb{N}$,

$$f_x(y) = \{ \text{recall definition } f_x \}$$

$$f_x(y) = \sum_{i=0}^{y} f_m(i) + f_m(y) \cdot \sum_{j=0}^{y-1} f_l(j)$$

$$= \{ \text{definition throughput distribution; distinguish cases } y = 0 \text{ and } y > 0 \}$$

$$= \begin{cases} f_l(y) \cdot \sum_{i=0}^{y} f_m(i) + f_m(y) \cdot F_l(y-1), & y > 0 \\ f_l(y) \cdot f_m(y), & y = 0 \end{cases}$$

The computation of the distribution function $F_m$ can be done in $u_m$ steps, just as the distribution function $F_l$ can be computed in $u_l$ steps. Therefore, the total computation of $f_x$ can be done in $\Theta(t)$ time, with $t$ equal to the maximum of $u_l$ and $u_m$.

On basis of the Convolution theorem we can compute the throughput density of the entire building block $B$ in $\Theta(n \log n)$ time, with $n$ equal to the smallest power of two that is at least twice as large as the maximum of the upper bounds of tasks $k$, $l$, $m$, and $n$.

3.4 Choice building block

The final building block we will consider in this chapter is the choice building block. The choice building block $B$ is depicted in Figure 10.

![Figure 10: Choice building block](image)

The building block $B$ consists of two tasks $s$ and $t$. Initiating building block $B$ results in either the execution of task $t$ or task $s$, with respective chances $\alpha$ and $1-\alpha$. When the selected task is completed, the building block itself is completed. We would like to compute throughput density $f_B$, given $f_s$ and $f_l$.

Let $y \in \mathbb{N}$,

$$f_B(y)$$
= { structure workflow net; introduce random variable I which determines whether task s or task t is executed }

\[ P((I = s \land s = y) \lor (I = t \land t = y)) \]

= \{ P(I = s) = \alpha; P(I = t) = 1 - \alpha; I \perp s \text{ and } I \perp t \text{ are independent } \}

\[ \alpha f_s(y) + (1 - \alpha)f_t(y) \]

From the last expression follows that we can compute \( f_B \) in \( \Theta(n) \) time, with \( n \) equal to the maximum of \( u_s \) and \( u_t \).

4. Computation of the overall throughput

4.1 Algorithm

In section 2 we have shown how proper workflow nets can be constructed with the building blocks choice, iteration, parallelism, and sequence. For each of these building blocks we know how to efficiently compute its throughput time, given the throughput times of its elements, i.e. the tasks. From here, it is a small step to compute the throughput time of the entire constructed workflow net.

Suppose we have constructed a workflow net \( W = (P_w, T_w, F_w) \) from the start net \( SN \) with \( n \) subsequent applications of the construction rule. Then, for \( n > 1 \) we can distinguish the intermediate workflow nets \( W_1, W_2...W_{n-1} \). \( W_1 \) is the result of the application of the construction rule using one of the building blocks on the start net \( S \); \( W \) results from the \( n \)th application of the construction rule on intermediate net \( W_{n-1} \). For each of the other intermediate nets \( W_i \) holds that it is the result from the \( i \)th application of the construction rule on intermediate net \( W_{i-1} \). Note that we can represent all intermediate workflow nets hierarchically in a derivation tree. It is, in fact, this derivation tree that we step through during our computation.

To compute the total throughput time of the constructed workflow net \( W \) we assume that the throughput density of each of its tasks is known. In addition, all probability distributions that have to do with choices or iterations are known too. Now we take the opposite direction of the construction route. Starting at the end, we consider the last application of the construction rule that leads from \( W_{n-1} \) to \( W \). We know which part of \( W_{n-1} \) is replaced by which specific building block. We call this part \( S_{n-1} \) and the building block \( B_{n-1} \). Recall that \( S_{n-1} \) is isomorphic with the start net. \( B_{n-1} \) is the part of
which we compute its throughput density $t_{n-1}$. This is feasible, as we know all relevant throughput densities, as well as the relevant probability distributions in $W$.

The resulting throughput density $t_{n-1}$ is a fair characterisation of the time that it takes $B_{n-1}$ to process a job. What is more, the building block can be seen as a detailed specification of the behaviour of $S_{n-1}$ that it has replaced. Therefore, the resulting throughput density is a fair characterisation of the time that it takes $S_{n-1}$ to process a job as well. When the only task in $S_{n-1}$ should have a throughput density that is equal to $t_{n-1}$, the total throughput time of $W_{n-1}$ would exactly be the same as that of $W$. $S_{n-1}$ can be seen as a "black box" for the applied building block. The task of computing the throughput time of $W$ has now become the task to compute the throughput time of $W_{n-1}$.

We can repeat this approach for each of the $n$ transformations. When we ensure that we follow the construction route in opposite direction, we can be confident that all relevant data is available to compute the throughput time of each start net that has been replaced. Finally, we end up with the start block $SN$. This block will have only one task of which its throughput density characterisation is exactly the throughput characterisation of the entire workflow net $W$. And this is exactly the throughput density we were looking for.

4.2 Numerical experience

To give an indication of the efficiency of the presented algorithms we have computed the throughput density of the workflow net depicted in Figure 4. As tool for these calculations we have used the mathematical software package Maple V® running on a Pentium 100 MHz computer. Each of the nine tasks in the workflow net has been modelled to behave in accordance with a distinct, arbitrary throughput density bounded by twenty time units. The probabilities $\alpha$ and $\beta$ of the model have been set on 0.3 and 0.1 respectively. For the computation of the throughput density of the iteration building block an inaccuracy ($\epsilon$) of 1 percent has been allowed.

The computation of the throughput density of the entire net under these circumstances took approximately 14 seconds. The eight Fast Fourier Transforms and two inverse Fast Fourier Transforms that are minimally needed for this calculation account for most of the duration, each transformation adding slightly over 1.1 seconds to the overall computing time. The computation time is likely to be reduced by the application of dedicated software instead of a mathematical package. A real
gain in computation time will probably be achieved when the time bound is set higher than twenty units.

5. Conclusion

In this paper we have demonstrated a way to efficiently compute a characterisation of the throughput time of a business process. Business processes have been modelled using a specific class of Petri nets. The logistic, general constructs that can be applied in these nets are sequence, parallelism, choice, and iteration. We have opened the possibility to model the dynamic behaviour of the separate tasks. Their execution time may vary within certain probability bounds for each job. In this way, the model of the business process is more realistic compared to the situation where constant task throughput times are used. Even more realism is added by allowing probability characteristics in choice and iteration constructs.

The scope of the presented approach is limited. We have only considered a quite quantitative and straightforward improvement target, i.e. the throughput time. Notwithstanding its importance we intend to extend the set of business improvement targets. Another limitation of our approach is that the impact of scarce resources on the throughput time is not taken into account. Right now, we are investigating an approach to incorporate the resource dimension. Also, the numerical efficiency of the algorithms can be improved. We have only demonstrated the feasible efficiency of computations on basis of generally available mathematical theory. Finally, it would be interesting to compare our results with a straightforward simulation approach.

The main subject of this paper, determining the throughput pattern of a business process, is not an end in itself. It is a prerequisite for improving a business process when the throughput time is at stake. When a more stable or shorter throughput time of a business process is required, it is of eminent importance that the begin situation can be established. Moreover, the effects of changes to the business process can only be evaluated when the throughput time characteristic of the new process can be determined.

On a more abstract level, the algorithms presented in this paper can be seen as some of the many algorithms that a business process redesign tool should incorporate. A tool that can present relevant analytical characteristics of a business process can be a valuable asset to the creative activity of redesigning it. In this way, redesign measures formerly only justified by "gut feeling" can be rationalised. Considering
the money, time and stakes involved with BPR we would embrace an increased rationality of the redesign process that may be acquired with the use of a redesign tool.
References


<table>
<thead>
<tr>
<th>Computing Science Reports</th>
<th>Department of Mathematics and Computing Science</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In this series appeared:</strong></td>
<td><strong>Eindhoven University of Technology</strong></td>
</tr>
<tr>
<td>96/01 M. Voorhoeve and T. Basten</td>
<td>Process Algebra with Autonomous Actions, p. 12.</td>
</tr>
<tr>
<td>96/02 P. de Bra and A. Aerts</td>
<td>Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12.</td>
</tr>
<tr>
<td>96/04 S. Mauw</td>
<td>Example specifications in phi-SDL.</td>
</tr>
<tr>
<td>96/06 W.M.P. van der Aalst and T. Basten</td>
<td>Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.</td>
</tr>
<tr>
<td>96/07 M. Voorhoeve</td>
<td>Structural Petri Net Equivalence, p. 16.</td>
</tr>
<tr>
<td>96/11 T. Laan</td>
<td>AUTOMATH and Pure Type Systems, p. 30.</td>
</tr>
<tr>
<td>96/12 F. Kamareddine and T. Laan</td>
<td>A Correspondence between Nuprl and the Ramified Theory of Types, p. 12.</td>
</tr>
<tr>
<td>96/13 T. Borghuis</td>
<td>Priorean Tense Logics in Modal Pure Type Systems, p. 61</td>
</tr>
<tr>
<td>96/14 S.H.J. Bos and M.A. Reniers</td>
<td>The $I^2C$-bus in Discrete-Time Process Algebra, p. 25.</td>
</tr>
<tr>
<td>96/15 M.A. Reniers and J.J. Vereijken</td>
<td>Completeness in Discrete-Time Process Algebra, p. 139.</td>
</tr>
<tr>
<td>96/17 E. Boiten and P. Hoogendijk</td>
<td>Nested collections and polytypism, p. 11.</td>
</tr>
<tr>
<td>96/18 P.D.V. van der Stok</td>
<td>Real-Time Distributed Concurrency Control Algorithms with mixed time constraints, p. 71.</td>
</tr>
<tr>
<td>96/19 M.A. Reniers</td>
<td>Static Semantics of Message Sequence Charts, p. 71</td>
</tr>
<tr>
<td>96/20 L. Feijs</td>
<td>Algebraic Specification and Simulation of Lazy Functional Programs in a concurrent Environment, p. 27.</td>
</tr>
<tr>
<td>96/23 W.M.P. van der Aalst</td>
<td>Structural Characterizations of sound workflow nets, p. 22.</td>
</tr>
<tr>
<td>96/24 M. Voorhoeve and W. van der Aalst</td>
<td>Conservative Adoption of Workflow, p.22</td>
</tr>
<tr>
<td>96/25 M. Vaccari and R.C. Backhouse</td>
<td>Deriving a systolic regular language recognizer, p. 28</td>
</tr>
<tr>
<td>97/06 M. Fransen</td>
<td>Tools for the Construction of Correct Programs: an Overview, p. 33.</td>
</tr>
<tr>
<td>97/07 J.C.M. Baeten and J.A. Bergstra</td>
<td>Bounded Stacks, Bags and Queues, p. 15.</td>
</tr>
</tbody>
</table>
When do datatypes commute? p. 35.

Communication Modeling - The Language/Action Perspective, p. 147.

Distributed real-time systems: a survey of applications and a general design model, p. 31.


Exploring the Process Dimension of Workflow Management, p. 56.

A computer checked algebraic verification of a distributed summation algorithm, p. 28

λP*: A Pure Type System for First Order Logic with Automated Theorem Proving, p. 35.

On the verification of Inter-organizational workflows, p. 23

Calculating a Round-Robin Scheduler, p. 23.


State Event Net Equivalence, p. 25

Deadlock Behaviour in Split and ST Bisimulation Semantics, p. 15.

Pair Algebras and Galois Connections, p. 14

Flat Fragments of CTL and CTL*: Separating the Expressive and Distinguishing Powers, p. 22.

Maintenance of the Union of Intervals on a Line Revisited, p. 10.

