Modeling of Time WED-flow

Rodrigo Alves Lima

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. João Eduardo Ferreira

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, novembro de 2017

Modeling of Time WED-flow

Esta é a versão original da dissertação elaborada pelo
candidato Rodrigo Alves Lima, tal como
submetida à Comissão Julgadora.

Resumo

LIMA, R. A. **Modelagem de WED-flow Temporal**. 2017. 120 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2017.

Organizações que buscam apoiar seus processos de negócio de forma mais flexível estão cada vez mais interessadas em substituir os sistemas de informação existentes centrados em dados por sistemas de informação cientes de processo (PAISs), nos quais a lógica de controle de fluxo dos processos é especificada em modelos executáveis separados do código da aplicação. Ainda que as abordagens tradicionais de modelagem de processos sejam amplamente utilizadas para o desenvolvimento de PAISs empresariais, muitas não possuem a flexibilidade necessária para: realizar reengenharia de processos de negócios; gerenciar dependências entre processos interativos e paralelos; e manter a base de código para tratar exceções e estados de processo imprevistos pequena e gerenciável. WED-flow, por sua vez, é uma abordagem de modelagem de processos transacional, baseada em eventos, e orientada a dados que atende a esses desafios. No entanto, WED-flow não pode modelar o comportamento e as restrições temporais dos processos sensíveis ao tempo encontrados em sistemas de tempo real. Para a maioria dos sistemas de tempo real, o não cumprimento de restrições de tempo pode resultar em catástrofes. Assim, a capacidade de decidir se um processo sensível ao tempo pode ou não cumprir seus prazos é essencial para ao menos aliviar seus efeitos colaterais potencialmente perigosos. Além disso, se processos sensíveis ao tempo competirem por recursos compartilhados, a detecção antecipada de que alguns deles não atenderão aos seus prazos pode aumentar a eficiência da política de alocação de recursos em uso. As principais contribuições deste trabalho são: (1) apresentar o modelo WED-flow Temporal, estendendo WED-flow com uma noção de tempo; (2) apresentar um método para mapear um modelo de processo WED-flow para uma rede de Petri – um formalismo gráfico e matemático para modelagem e análise de sistemas concorrentes; (3) e apresentar um modelo de rede Petri dependente de tempo adequado para descrever a semântica temporal dos modelos de processo de WED-flow Temporal. Assim, podemos verificar a lógica de controle de fluxo dos modelos de processo WED-flow através de suas redes de Petri equivalentes, assim como verificar sua pontualidade. Como um exemplo de um sistema de tempo real modelado usando WED-flow, apresentamos SISAUT – um sistema de gerenciamento de autópsias que coordena processos interativos, paralelos e sensíveis ao tempo para coletar e processar órgãos para projetos de pesquisa.

Palavras-chave: WED-flow. WED-flow Temporal, sistemas de tempo real, rede de Petri, rede de Petri dependente de tempo.

Abstract

LIMA, R. A. . 2010. 120 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2017.

Organizations seeking to support their business processes in a more flexible way are increasingly interested in replacing existing information systems centered on data with process-aware information systems (PAISs), in which the control-flow logic of processes is specified in executable models separate from the application code. Even though traditional process modeling approaches have been largely used to build enterprise PAISs, most lack the flexibility needed to: perform business process reengineering; manage dependencies among interacting, parallel processes; and keep the code base for handling exceptions and unforeseen process states small and manageable. WED-flow, in turn, is a transactional, event-based, and data-driven process modeling approach that addresses these challenges. However, WED-flow cannot model the temporal behavior and constraints of time-critical processes found in real-time systems. For most real-time systems, failing to meet time constraints could result in catastrophic damage. Hence, the ability to decide whether or not a time-critical process can meet its deadlines is essential to at least alleviate potentially hazardous side effects. Moreover, if time-critical processes compete for shared resources, early detecting that some of them are likely to miss their deadlines could increase the efficiency of the resource allocation policy in use. The main contributions of this work are: (1) to present Time WED-flow, extending WED-flow with the notion of time; (2) to present a method for mapping a WED-flow process model to a Petri net – a graphical and mathematical formalism for modeling and reasoning about the functional behavior of concurrent systems; (3) and to present a time dependent Petri net model suitable for describing the temporal semantics of Time WED-flow process models. Thus, we can check the control-flow logic of WED-flow process models through their equivalent Petri nets, and check their timeliness as well. As an example of a real-time system modeled using WED-flow, we present SISAUT – an autopsy management system that coordinates interacting, parallel, time-critical processes to collect and process organs for research projects.

Keywords: WED-flow, Time WED-flow, real-time systems, Petri net, time dependent Petri net.

Contents

Chapter 1

Introduction

Organizations seeking to support their business processes in a more flexible way are increasingly interested in replacing existing information systems centered on data with process-aware information systems (hereinafter referred to as PAISs) [DvdAtH], in which the control-flow logic of processes is specified in executable models separate from the application code [RW12]. Reichert and Weber identified four flexibility needs of business processes to be met by PAISs: variability, evolution, looseness, and adaptation [RW12]. In the following, we briefly describe each of these needs:

- *Variability* is needed to model variants of a single business process. For example, consider a car rental business process. Distinct state traffic laws might apply depending on the location selected to pick up the car (e.g., whether or not an international driver's license is valid), and this business process is expected to handle them all.

- *Looseness* is needed to support a business process dependent on human knowledge. For example, consider a hospital's medical care scheduling system. After a patient goes through the triage, the selection of each next activity to be performed (e.g., physical examination, collection of blood sample, etc.) is dependent on the medical staff.

- *Evolution* can be understood as the need to update the control-flow logic or the application code of a business process. For example, consider a travel agency's business process for booking flight tickets. At some point, one might decide to extend this business process with a service for booking hotel rooms.

- *Adaptation* can be understood as the need to handle exceptions of a business process. For example, consider an e-commerce's online shopping business process. If a client's credit card is denied at checkout, another option of payment method should be promptly offered.

In the following, we briefly describe four of the most popular process modeling approaches: graph-based, rule-based, constraint-based, and data-driven.

- A *graph-based* process model is a graph of activity nodes connected to each other and to control-flow primitives [LS07]. Graph-based process modeling approaches are suitable for modeling repetitive and predictable processes. BPMN (Business Process Model and Notation) [Whi04] and YAWL (Yet Another Workflow Language) [VDATH05] are well-known examples of graph-based process modeling approaches.

- A *rule-based* process model is composed of a set of so-called ECA (Event-Condition-Action) rules [LS07]. Such a rule is of the form `on [event] if [condition] do [activity]`, which means that `activity` is performed when `event` happens if `condition` is satisfied. Rule-based process modeling approaches are particularly appropriate for modeling rapidly evolving processes. Active database systems have been commonly used for enacting rule-based process models [KRSR96] [BJ94].

- A *constraint-based* process model is composed of a set of constraints preventing undesired execution behaviors. Constraint-based process modeling approaches are suitable for modeling processes dependent on human knowledge. Dynamic Condition Response Graphs [HM10] and ConDec [PVdA06] are examples of constraint-based process modeling approaches.

- A *data-driven* process model describes the lifecycle of process entities represented as data objects. Business Artifacts [CH09] and Case Handling [VdAWG05] are examples of data-driven process modeling approaches.

Even though traditional process modeling approaches have been largely used to build enterprise PAISs, most lack the flexibility needed to perform business process reengineering [Att04] and fail to keep the code base for handling exceptions and unforeseen process states small and manageable. Furthermore, software engineers often rely on message queues and other middleware components to manage dependencies among interacting, parallel processes, which not only add another layer of complexity to the system but also shatter its execution history across many data repositories.

WED-flow, in turn, is a transactional, event-based, and data-driven process modeling approach that addresses these challenges [FBTP12] [FTK+]. In summary, WED-flow can keep track of the communication between interacting, parallel processes through shared data states [FTK+]. Also, by harnessing the ACID properties of classical database transactions [GR92], WED-flow has a simple yet powerful transactional recovery mechanism based on its well-defined failure semantics [FBTP12].

However, WED-flow cannot model the temporal behavior and constraints of time-critical processes found in real-time systems [Lap04], which are expected to not only produce correct results but also to meet pre-specified time constraints. The correctness of a time-critical process, therefore, depends on its outputs as well as on the time these outputs are produced. Typical examples of real-time systems include flight control, power plant control, multimedia streaming, and anti-lock braking systems. A power plant control system, for example, should be expected to take some action to cool down a nuclear reactor when its temperature is above a safe operating level. Furthermore, failing to take this action within a short period of time could result in hazardous side effects. In fact, for most real-time systems, failing to meet time constraints could result in catastrophic damage. Hence, the ability to decide whether or not a time-critical process can meet its deadlines is essential to at least alleviate potentially hazardous side effects. Moreover, if time-critical processes compete for shared resources, early detecting that some of them are likely to miss their deadlines could increase the efficiency of the resource allocation policy in use.

Formal verification methods have been used to check the correctness of both functional (e.g., decide whether or not a desired state can be reached) and temporal (e.g., decide whether or not a deadline can be met) behaviors of real-time systems [HM96]. In particular, model checking [CGP99] is a formal verification method that has been successfully used for verifying real-time systems [ACD90]. Model checking techniques evaluate models of concurrent systems rather than their actual executable code, though, and these models should be specified in terms of a mathematical notation (e.g., Petri net [Mur89]) or formal specification language (e.g., TLA+ [Lam02] and Promela [Hol97]). Further, these models should capture the semantics needed to establish the correctness of those concurrent systems while abstracting away unnecessary details for their checking to be tractable.

The main contributions of this work are (1) to present Time WED-flow, extending WED-flow with the notion of time; (2) to present a method for mapping a WED-flow process model to a Petri net – a graphical and mathematical formalism for modeling and reasoning about the functional behavior of concurrent systems; (3) and to present a time dependent Petri net model suitable for describing the temporal semantics of Time WED-flow process models. Thus, we can check the control-flow logic of WED-flow process models through their equivalent Petri nets, and check their timeliness as well. As an example of a real-time system modeled using WED-flow, we present SISAUT – an autopsy management system that coordinates interacting, parallel, time-critical processes to collect and process organs for research projects.

In Chapter 2, we present the fundamentals of WED-flow and a method for mapping a WED-flow process model to a Petri net. We then extend WED-flow with the notion of time and present

a time dependent Petri net model suitable for describing the temporal semantics of Time WED-flow process models in Chapter 3. Next, we present SISAUT as an example of a real-time system modeled using WED-flow in Chapter 4. Finally, we draw our conclusions in Chapter 5.

Chapter 2

The WED-flow Process Modeling Approach

WED-flow (Work, Event, Data-flow) is a long-running transaction model in which data states are first-class citizens and the control-flow logic of process models is specified by rules. A WED-flow process instance can be seen as a state transition system: starting from an initial state, state transitions are continuously applied until a final state is reached. Such a state transition is implemented with a classical database transaction [GR92], and thus can be seen as a *saga step* [GMS87].

Each WED-flow process instance has a data state (*WED-state*), which can be seen as a dictionary mapping pre-specified attributes of interest (*WED-attributes*) to values. So-called *WED-triggers* are rules bonding logical predicates (*WED-conditions*) together with state transitions (*WED-transitions*). A special logical predicate (*initial WED-condition*) validates the first WED-state created for a WED-flow process instance, and another special logical predicate (*final WED-condition*) terminates WED-flow process instances.

2.1 Definitions

Unless explicitly noted, the following definitions are based on the ones presented in [FBTP12].

Definition 2.1.1 (WED-attribute). *A WED-attribute is an attribute of interest of the process being modeled. The WED-attributes defined for a process $\mathcal{P}$ are formally given by the n-tuple $A = (a_1, ..., a_n)$. Also, a domain has to be assigned to each WED-attribute of A. For example, if a_k is a boolean WED-attribute, $1 \leqslant k \leqslant n$, $domain(a_k) = \{true, false\}$.*

Definition 2.1.2 (WED-state). *A WED-state is a valuation of the WED-attributes of the process*

being modeled. A WED-state of $\mathcal{P}$ is formally given by an n-tuple $(v_1, ..., v_n)$ such that, for all $1 \leqslant i \leqslant n$, $v_i \in domain(a_i)$. The set of all possible WED-states of $\mathcal{P}$ is denoted by S.

Definition 2.1.3 (WED-condition). *A WED-condition is a logical predicate defined over the WED-attributes of the process being modeled. A WED-condition of $\mathcal{P}$ is formally given by a function from S to the set of boolean values. The set of WED-conditions defined for $\mathcal{P}$ is denoted by C. Also, a WED-state $s \in S$ is said to satisfy a WED-condition $c \in C$ if and only if $c(s) = true$.*

Definition 2.1.4 (WED-transition). *A WED-transition is a data state transition function of the process being modeled. A WED-transition of $\mathcal{P}$ is formally given by a function from S to S. The set of WED-transitions defined for $\mathcal{P}$ is denoted by U. Also, WED-transitions are implemented with ACID transactions [GR92], which ensure the serializability of concurrent WED-transitions.*

Definition 2.1.5 (WED-trigger). *A WED-trigger of $\mathcal{P}$ is formally given by a 2-tuple (c, u), where $c \in C$ and $u \in U$. The set of WED-triggers defined for $\mathcal{P}$ is denoted by G.*

Definition 2.1.6 (WED-flow process model). *A WED-flow process model of $\mathcal{P}$ is formally given by a 3-tuple $WF = (G, c_i, c_f)$, where G is its finite set of WED-triggers, $c_i \in C$ is its initial WED-condition, and $c_f \in C$ is its final WED-condition. The initial WED-condition c_i tests if the first WED-state created for each instance of WF is valid. If c_i is satisfied, each WED-trigger of G is then evaluated with respect to the initial WED-state. For example, consider a WED-trigger $(c, u) \in G$. If c is satisfied by the initial WED-state, u is triggered. Furthermore, if u is triggered, it eventually creates a new WED-state, for which each WED-trigger of G is evaluated once more, and so on. The final WED-condition c_f tests each newly created WED-state as well. If c_f is satisfied, though, an instance of WF is promptly terminated. Also, an instance of WF is said to be properly terminated if no WED-transition is running when c_f is satisfied and its final WED-state does not trigger any other WED-transition.*

WED-states satisfying the initial and final WED-conditions are called *initial* and *final WED-states*, respectively. WED-states that trigger the execution of at least one WED-transition are said to be *transaction-consistent*. It may be the case, however, that a non-final WED-state does not trigger the execution of any WED-transition. If at least one WED-transition is running when such a WED-state is created, it is also said to be transaction-consistent. Otherwise, it is said to be *inconsistent*.

Despite having only a few primitives and being governed by two simple rules, under certain assumptions, Petri nets can precisely and unambiguously describe WED-flow semantics due to

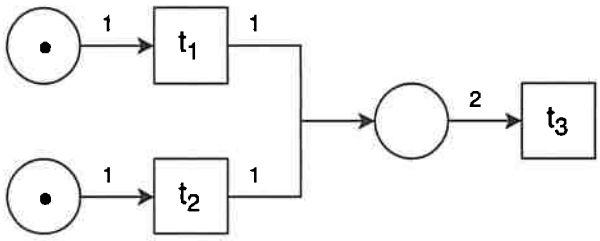
their explicit representation of states and simple yet powerful model of concurrency. Furthermore, one can readily grasp the control-flow logic of WED-flow process models from the plain graphical representation of their equivalent Petri nets.

2.2 The Petri Net Model

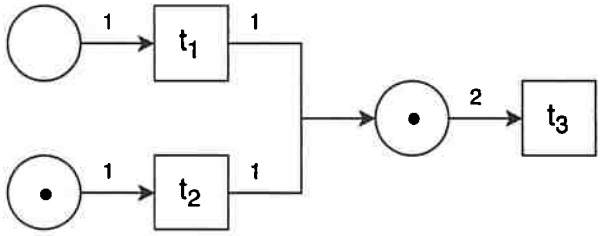
Proposed by Carl Adam Petri in 1962 [Pet62], Petri net is a graphical and mathematical formalism for modeling and reasoning about the functional behavior of concurrent systems [Mur89]. A Petri net is a particular type of graph extended with a state. The underlying graph of a Petri net is directed, weighted, and bipartite. A vertex can be of one of two types: a *place* or a *transition*. *Arcs* are either from a place to a transition or from a transition to a place. If an arc connects a place p to a transition t, p is said to be an *input place* of t. If an arc connects a transition t to a place p, though, p is said to be an *output place* of t. The state of a Petri net is called a *marking*. A marking assigns a number of *tokens* to each place of that Petri net. Transitions are commonly interpreted as being events, so their input and output places are interpreted as being pre- and post-conditions of these events, respectively. The presence of a token in a place thus means that the condition (or part of it) represented by that place holds true. Graphically, places are represented as circles and transitions as rectangles. Arcs are labeled with their weights (except for arcs of weight 1, which are usually not labeled). If a marking assigns k tokens to a place, k black dots are drawn inside the circle representing that place.

Definition 2.2.1 (Petri net). *A Petri net is formally given by a 5-tuple* (P, T, F, w, M_0), *where* $P = \{p_1, ..., p_n\}$ *is its finite set of places,* $T = \{t_1, ..., t_m\}$ *is its finite set of transitions,* $F \subseteq (P \times T) \cup (T \times P)$ *is its finite set of arcs,* $w : F \to \{1, 2, 3, ...\}$ *is the weight function of its arcs, and* $M_0 : P \to \{0, 1, 2, ...\}$ *is its initial marking.*

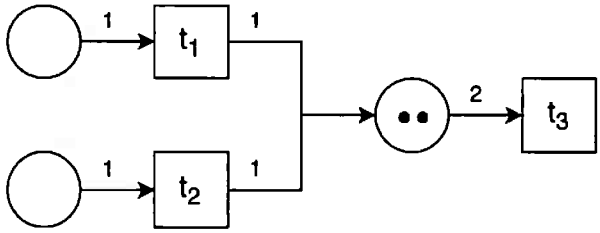
A transition is said to be *enabled* in a marking if and only if in each of its input places there are a number of tokens greater than or equal to the weight of their connecting arc. If a transition is enabled, it can *fire*. Firing a transition consumes and produces tokens according to the following rule: from each input place of that transition are removed a number of tokens equal to the weight of their connecting arc, and to each output place of that transition are added a number of tokens equal to the weight of their connecting arc. At any given time, only one transition can fire in a Petri net, and firings of transitions are considered to be instantaneous.



(a) Both t_1 and t_2 are enabled in this marking, and the choice of which transition fires first is non-deterministic.



(b) t_1 fires, removing one token from its input place and adding one token to its output place.



(c) t_2 fires and t_3 becomes enabled as a result.

Figure 2.1: A sequence of firings of transitions in a Petri net.

2.3 Mapping p-WED-flow Process Models to Petri Nets

Now we map a subset of WED-flow to the Petri net model. This subset is derived from the programming style used in the development of the PAIS presented in Chapter 4, and from now on is referred to simply as *p-WED-flow*. In short, p-WED-flow captures the control-flow logic of WED-flow process models while abstracting away their data-flow perspective.

First, let $WF = (G, c_i, c_f)$ be a p-WED-flow process model, where G is its finite set of WED-triggers, $c_i \in C$ is its initial WED-condition, and $c_f \in C$ is its final WED-condition. Also, let $A = (a_1, ..., a_n)$ be its tuple of WED-attributes, $C = \{c_1, ..., c_n\}$ be its set of WED-conditions, and $U = \{u_1, ..., u_n\}$ be its set of WED-transitions.

For all $1 \leqslant k \leqslant n$, we make the following assumptions:

1. $domain(a_k) = \{true, false\}$.

2. The value of a_k is initialized with $false$.

3. The initial WED-condition c_i tests if the value of a_k is $false$.

4. The value of a_k is $true$ if and only if u_k has already been executed.

5. c_k is a conjunction of one or more evaluations of the values of WED-attributes.

6. Besides c_i, c_k is the only WED-condition to test if the value of a_k is $false$.

We also assume that $G = \{(c_k, u_k) \in C \times U \mid \text{for all } 1 \leqslant k \leqslant n\}$. s_0 denotes the single initial WED-state of WF, in which the value of all WED-attributes is $false$.

Now let $PN = (P, T, F, w, M_0)$ be a Petri net, where:

- $P = \{p_k \mid \text{for all } 1 \leqslant k \leqslant n\} \cup \{p_f\}$ is its finite set of places. p_k, $1 \leqslant k \leqslant n$, represents the WED-condition c_k of WF, and p_f represents the final WED-condition c_f of WF.

- $T = \{t_k \mid \text{for all } 1 \leqslant k \leqslant n\} \cup \{t_f\}$ is its finite set of transitions. t_k, $1 \leqslant k \leqslant n$, represents the WED-transition u_k of WF, and t_f is called the *final transition.*

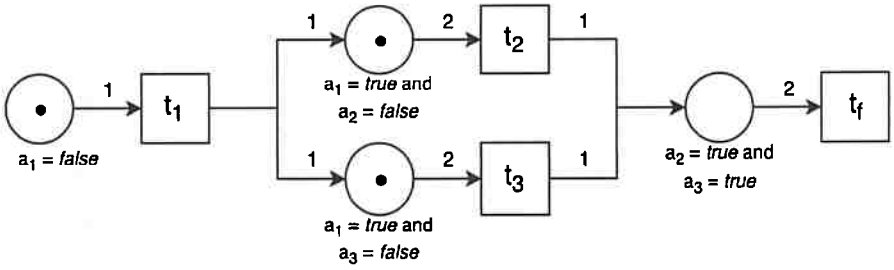
- $F = \{(p_k, t_k) \in P \times T \mid \text{for all } 1 \leqslant k \leqslant n\} \cup \{(t_k, p_j) \in T \times P \mid$ the WED-condition c_j of WF tests if the value of the WED-attribute a_k is $true$, for all $1 \leqslant k \leqslant n$, for all $1 \leqslant j \leqslant n\} \cup \{(p_f, t_f)\}$ is its set of arcs.

- w is the weight function of its arcs. Here we assume that the cardinality of a place gives the number of WED-attribute evaluations of the WED-condition it represents. Then, for all $1 \leqslant k \leqslant n$, $w((p_k, t_k)) = |p_k|$, and for all $1 \leqslant j \leqslant n$, $w((t_k, p_j)) = 1$, if $(t_k, p_j) \in F$. Also, $w((p_f, t_f)) = |p_f|$.

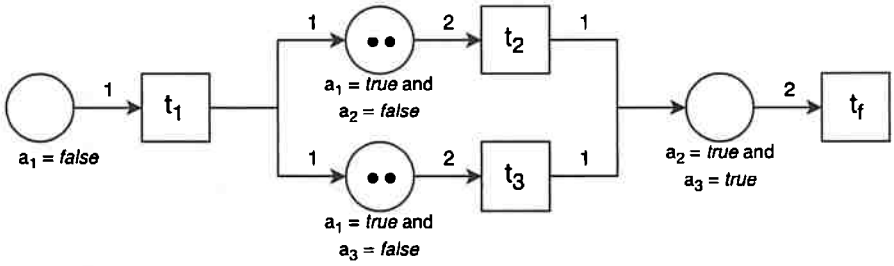
- M_0 is its initial marking. In M_0, there is exactly one token in each place, except for p_f, in which there are no tokens.

By the construction of PN and from the assumptions made, it straightforwardly follows that a WED-trigger of WF is represented as a transition and an input place in PN. The input place represents the WED-condition and the transition represents the WED-transition of that WED-trigger, respectively. The weight of the arc connecting an input place representing a WED-condition to a transition representing a WED-transition is given by the number of WED-attribute evaluations of that WED-condition, and therefore each token put in that input place represents that some WED-attribute evaluation of that WED-condition holds true. Enabling a transition t_k, $1 \leqslant k \leqslant n$, is then interpreted as triggering the WED-transition u_k, and firing t_k is interpreted as terminating u_k. We thus say that t_k and u_k are correspondent. The place p_f represents the final WED-condition c_f, and since p_f is an input place of the final transition t_f, firing t_f is interpreted as terminating an instance of WF.

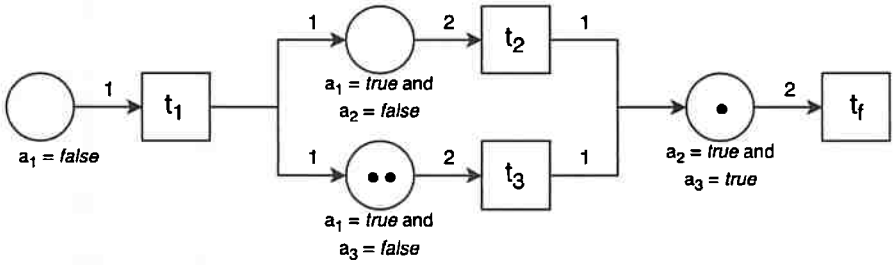
Now we prove that the Petri net PN is equivalent to the p-WED-flow WF.



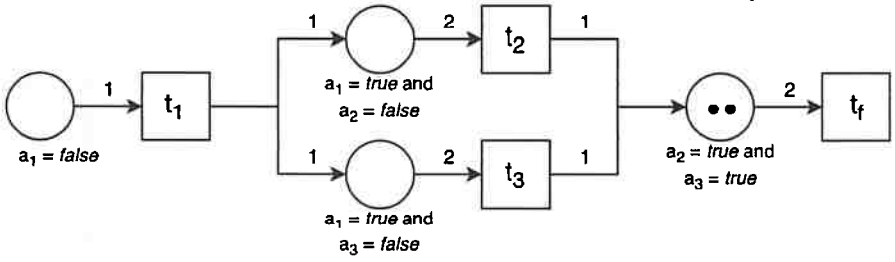
(a) In the initial marking, only t_1 is enabled.



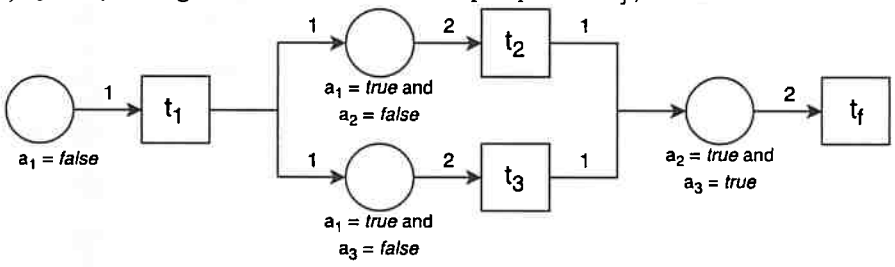
(b) t_1 fires, adding one token to the input place of t_2 and one token to the input place of t_3.



(c) t_2 fires first, adding one token to the input place of t_f.



(d) t_3 fires, adding one more token to the input place of t_f, which becomes enabled.



(e) t_f fires, and in the resulting marking there are no tokens in any place. This instance is said to be properly terminated.

Figure 2.2: A sequence of firings of transitions in a Petri net equivalent to a p-WED-flow.

Lemma 2.3.1 (There exists a correspondence between the initial marking M_0 and the initial WED-state s_0.). *From assumptions (2), (5), and (6), only one WED-attribute evaluation of each*

WED-condition c_k, $1 \leqslant k \leqslant n$, is satisfied by the initial WED-state s_0. Also, no WED-attribute evaluation of the final WED-condition c_f is satisfied by s_0.

By the construction of PN, in M_0 there is exactly one token in each place p_k, $1 \leqslant k \leqslant n$, and there are no tokens in p_f.

Lemma 2.3.2 (There exists a bijection between the set of transitions enabled for the initial marking M_0 and the set of WED-transitions triggered by the initial WED-state s_0.). *From assumption (6), for all $1 \leqslant k \leqslant n$, c_k is the only WED-condition besides the initial WED-condition c_i that compares the value of a_k to $false$. From assumption (2), in s_0 the value of all WED-attributes is $false$. Therefore, s_0 triggers the WED-transition u_k if and only if c_k has only one WED-attribute evaluation, which is the comparison between the value of a_k and $false$.*

By the construction of PN, for all $1 \leqslant k \leqslant n$, in M_0 there is exactly one token in each place p_k, and there are no tokens in p_f. Therefore, a transition t_k is enabled in M_0 if and only if the weight of the arc connecting p_k to t_k is 1, i.e., c_k has only one WED-attribute evaluation, which is the comparison between the value of a_k and $false$.

Lemma 2.3.3 (Assume that a marking M is correspondent to a WED-state s. The resulting marking after a transition t_k, $1 \leqslant k \leqslant n$, fires is correspondent to the WED-state created after the WED-transition u_k terminates.). *From assumptions (2), (4), and (6), u_k creates a new WED-state in which the value of the WED-attribute a_k is updated from $false$ to $true$.*

By the construction of PN, firing t_k results in a new marking in which tokens are added to places representing WED-conditions that test if the value of the WED-attribute a_k is true.

Lemma 2.3.4 (Assume that there exists a bijection between the set of transitions enabled for a marking M and the set of WED-transitions running for a WED-state s. Then, there exists a bijection between the set of transitions that become enabled after a transition t_k, $1 \leqslant k \leqslant n$, fires and the set of WED-transitions triggered after the WED-transition u_k terminates.). *From assumptions (2), (4), (5), and (6), for $1 \leqslant j \leqslant n$, a WED-transition u_j is triggered after u_k terminates if and only if a_k = true was the single WED-attribute evaluation not being satisfied in the WED-condition c_j for the last WED-state.*

By the construction of PN, for $1 \leqslant j \leqslant n$, firing a transition t_k puts a token in place p_j if and only if the WED-condition c_j tests if the value of the WED-attribute a_k is true. Therefore, a transition t_j becomes enabled after t_k fires if and only if a_k = true was the single WED-attribute evaluation not being satisfied in the WED-condition c_j for the last WED-state.

Lemma 2.3.5 (In PN, an enabled transition necessarily fires.). *Each place of PN is an input place of at most one transition, and hence there are no conflicts.*

Theorem 2.3.1 (PN is equivalent to WF.). *Using Lemmas 2.3.1 and 2.3.2 as base case, Lemmas 2.3.3 and 2.3.4 as induction step, and Lemma 2.3.5, it follows from mathematical induction that there exists a bijection between the set of all possible sequences of firings of transitions from M_0 and the set of all possible execution paths from s_0.*

2.4 Checking the Control-Flow Logic of p-WED-flow Process Models

Specifying the control-flow logic of a process can be an error-prone task, especially when it comes to interacting, parallel processes with a large number of tasks. Therefore, checking WED-flow process models is essential for the development of reliable PAISs. In order to check a p-WED-flow process model, we analyze its equivalent Petri net.

Reachability is the fundamental property of Petri nets for reasoning about the functional behavior of concurrent systems. Let (P, T, F, w, M_0) be a Petri net. A sequence of firings of transitions results in a sequence of markings. A marking M_n is said to be reachable from the initial marking M_0 if there exists a sequence $\sigma = (t_1, ..., t_n)$ of firings of transitions such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} ...M_{n-1} \xrightarrow{t_n} M_n$. *Reachability tree* is a data structure used for the reachability analysis of Petri nets. The root node of a reachability tree represents the initial marking, and each child node recursively represents a marking reachable from the marking represented by its parent node through the firing of a single transition.

From the assumptions made about a p-WED-flow process model, each WED-transition is executed at most once, and therefore the reachability tree of its equivalent Petri net is finite. Also, by the construction of its equivalent Petri net, it straightforwardly follows that no token is put in an input place of a transition that have been already fired, and that firing the final transition represents the termination of a p-WED-flow process instance. Furthermore, here we assume that a p-WED-flow process instance is only considered to be properly terminated if all WED-transitions have been already triggered and terminated for that instance when the final WED-state is reached. Thus, in order to check if every instance of a p-WED-flow process model properly terminates, we should check if each leaf node of the reachability tree of its equivalent Petri net represents a marking in which there are no tokens in any place.

Chapter 3

The Time WED-flow Model

It is fair to say that most real-world processes managed by information systems have a final deadline. For example, an airline's business process for booking flight tickets should be expected to take at most a few tens of minutes to complete, so reserved seats could be released in case the client is unable to pay for the order within the final deadline. A college application process, in turn, could take a few months to complete. Therefore, in this Chapter we extend the WED-flow process modeling approach with the notion of time by specifying a final deadline for each created WED-flow process instance. From now on, such a time dependent WED-flow is called *Time WED-flow.*

Here we use WED-SQL [PSR17] – the standard implementation of the WED-flow process modeling approach. WED-SQL is a relational framework built on top of the PostgreSQL database management system [pos17]. PAISs implemented using WED-SQL are distributed: a single transaction manager called *WED-server* coordinates the execution of WED-flow process instances, and external services hosted on worker nodes called *WED-workers* perform their tasks.

The final deadline of a Time WED-flow process instance is a time value specified when it is created, which could even be a function of the value of one of its time WED-attributes.

```
create wed-instance as default values
       with timeout now() + interval '4 hours';
create wed-instance as (name, start_time)
       values ('John Doe', '2017-01-01 12:00')
```

```
with timeout start_time + interval '30 minutes';
```

Listing 3.1: Specifying the final deadline of WED-flow process instances in WED-SQL. The first WED-flow process instance created cannot take more than 4 hours from the time it is initialized to complete. The second one cannot take more than 30 minutes from the value of the WED-attribute *start_time* to complete.

3.1 Handling Timeouts

The WED-server periodically checks the final deadline of each running WED-flow process instance. As soon as the WED-server detects that a Time WED-flow process instance missed its final deadline, it (1) aborts instances of WED-transitions running for that Time WED-flow process instance, (2) promptly interrupts their associated instances of external services running on WED-workers, and (3) throws a timeout exception for that Time WED-flow process instance.

In WED-SQL, aborting WED-transitions is a trivial and efficient operation because they are implemented with PostgreSQL's transactions. Also, WED-workers are connected to the WED-server through a PostgreSQL's communication channel, which could then be used for sending and catching signals of needed interruptions. Timeout exceptions, in turn, are handled through the WED-flow transactional backward recovery mechanism presented in [FBTP12]. In summary, each WED-transition might be associated with a *compensation step* that semantically compensates it and performs needed corrective actions. Furthermore, a detailed execution history of each WED-flow process instance is kept by the WED-server, so those compensation steps can be triggered in the reverse order their correspondent WED-transitions were terminated until a so-called *stop WED-condition*, which can be seen as a dynamic breakpoint, or the initial WED-condition is satisfied. In Algorithm 1, we reproduce an excerpt of the WED-flow backward recovery algorithm presented in [FBTP12].

Now consider an airline's business process for booking flight tickets. A certain client selected a flight and reserved seats, but could not pay for the order within the final deadline. Two WED-transitions were successfully terminated, in the following order: *select_flight* and *reserve_seats*. Also, the WED-transition *pay_order* was running when the final deadline was missed. As a result:

1. The instance of the WED-transition *pay_order* running for this Time WED-flow process instance is aborted.

Input: WED-flow process instance i, initial WED-condition c_i, WED-condition c_{stop}.
Output: *true*, if successfully recovered. *false*, otherwise.
$s \leftarrow$ WED-state of i;
while *s does not satisfy* c_{stop} *and s does not satisfy* $c_{initial}$ **do**
  $t \leftarrow$ WED-transition that created s;
  if t^{-1} *is defined* **then**
    $s \leftarrow t^{-1}(s)$;
  else
    return *false*;
  end
end
return *true*;

Algorithm 1: WED-flow Backward Recovery Algorithm

2. The instance of the external payment processing service associated with that aborted WED-transition is interrupted.

3. A timeout exception is thrown for this Time WED-flow process instance. The first compensation step to be triggered is the one associated with the WED-transition *reserve_seats*. Such a compensation step should release reserved seats. When this compensation step terminates, the compensation step of the WED-transition *select_flight* is triggered. Actually, there is no need to compensate *select_flight* because it has no side effects. In this case, a stop WED-condition should be satisfied after the compensation step of *reserve_seats* created a new WED-state.

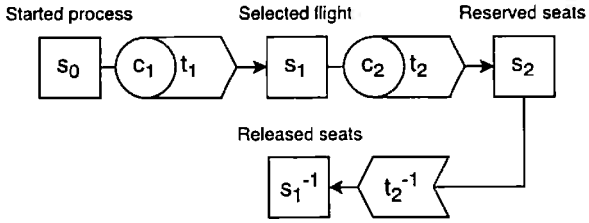


Figure 3.1: t_1, t_2, and t_3 represents *select_flight*, *reserve_seats*, and *pay_order*, respectively. t_2^{-1}, in turn, represents the compensation step of *reserve_seats*.

In order to check the timeliness of a Time WED-flow process instance, we need an estimate of the time it takes to execute each WED-transition, but since no assumption is made on the time it takes for a transition to fire, classic Petri nets cannot model the temporal behavior of Time WED-flow process models. Time dependent Petri net models, in turn, extend Petri nets with the notion of time, and mainly differ from each other on the primitive to which time is assigned (place, transition, or token) and on the nature of that time (deterministic, interval, or stochastic) [Wan98].

3.2 The Time Petri Net Model

Time Petri Net (hereinafter referred to as TPN) is a time dependent Petri net model that assigns a time interval to each transition. A transition, then, can only fire within this time interval once it is enabled. For example, suppose that a time interval $[d_{min}, d_{max}]$, where $d_{min}, d_{max} \in \mathbb{R}_+$ and $d_{min} \leqslant d_{max}$, is assigned to a transition t. d_{min} and d_{max} are called *earliest firing time* and *latest firing time* of t, respectively. Unless t is disabled by the firing of another transition, t must not fire before d_{min} units of time have elapsed since it was enabled, and t must fire no later than d_{max} units of time after it was enabled [PZ13].

Definition 3.2.1 (Time Petri Net). *A TPN is formally given by a 6-tuple (P, T, F, w, M_0, I) such that the 5-tuple (P, T, F, w, M_0) is a classic Petri net and $I : T \to \mathbb{R}_+ \times \mathbb{R}_+$ is its interval function. For each $t \in T$, $I(t) = (eft(t), lft(t))$, where $eft(t)$ and $lft(t)$ are the earliest and latest firing times of t, respectively.*

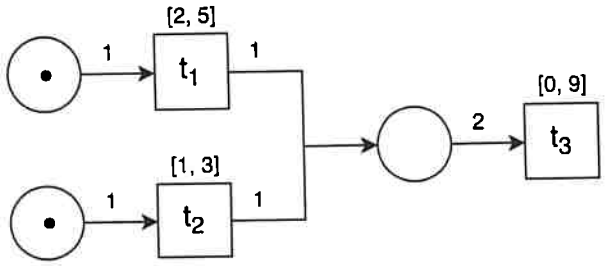
Besides the number of tokens in each place, the state of a TPN comprises the time elapsed since each transition was last enabled.

Definition 3.2.2 (State of a Time Petri Net). *The state of a TPN (P, T, F, w, M_0, I) is given by a 2-tuple (M, h), where $M : P \to \{0, 1, 2, ...\}$ is called a place-marking and $h : T \to R_+ \cup \{\#\}$ is called atransition-marking. For each $t \in T$, $h(t)$ gives the time elapsed since t was last enabled, if it is enabled. $h(t) = \#$, otherwise.*

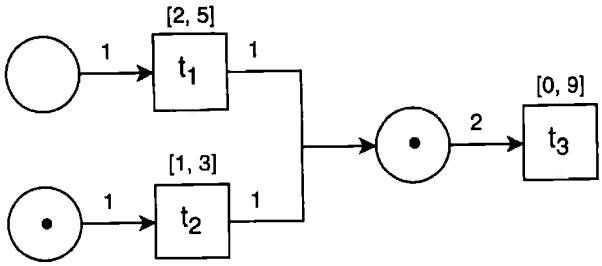
Now consider a TPN (P, T, F, w, M_0, I), whose state is given by a 2-tuple (M, h). A transition $t \in T$ can fire in this state if and only if t is enabled in M and $eft(t) \leqslant h(t) \leqslant lft(t)$. In this case, t is said to be *time-enabled*. If t fires, besides the usual changes in the place-marking M, the transition-marking h has to be updated with the transitions that become enabled and disabled as well. At any given time, only one transition can fire in a TPN, and firings of transitions are considered to be instantaneous. Furthermore, the state of this TPN can also change through the elapse of time. In this case, for each $t \in T$ enabled in M, the elapsed time δ is added to $h(t)$, given that $h(t) + \delta \leqslant lft(t)$.

3.3 Checking the Timeliness of Time p-WED-flow Process Instances

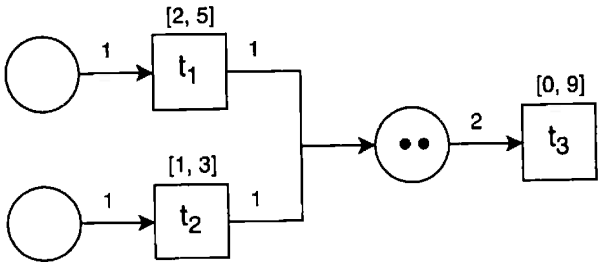
For most real-time systems, failing to meet time constraints could result in catastrophic damage. Hence, the ability to decide whether or not a time-critical process can meet its deadlines is essential



(a) In the initial state, both t_1 and t_2 are enabled but not time-enabled. $h(t_1) = 0$, $h(t_2) = 0$, and $h(t_3) = \#$, where h is the transition-marking of this TPN.



(b) After 2 seconds, both t_1 and t_2 are time-enabled, and t_1 fires. Now $h(t_1) = \#$, $h(t_2) = 2$, and $h(t_3) = \#$.



(c) 1 second later, t_2 fires and t_3 becomes time-enabled as a result. Now $h(t_1) = \#$, $h(t_2) = \#$, and $h(t_3) = 0$.

Figure 3.2: A sequence of firings of transitions in a TPN. The time interval assigned to each transition is shown on top of each rectangle.

to at least alleviate potentially hazardous side effects. Moreover, if time-critical processes compete for shared resources, early detecting that some of them are likely to miss their deadlines could increase the efficiency of the resource allocation policy in use. Therefore, checking the timeliness of Time WED-flow process instances is essential for the development of safe and efficient PAISs. Here we deal with p-WED-flow process models as well, and in order to check the timeliness of a Time p-WED-flow process instance, we analyze its equivalent TPN. The structure of this TPN is given by the method for mapping a p-WED-flow process model to an equivalent Petri net presented in Chapter 2, and its interval function gives best- and worst-case estimates of the time it takes to execute each WED-transition. Such an estimate could be set by a domain specialist or calculated by a statistical analysis of the logs of previous executions. For the timeliness check of a Time p-WED-flow process instance, we need to calculate lower and upper bounds for the time to take its current WED-state to a final WED-state, i.e., we need to calculate minimum and maximum distances of time from the state correspondent to its current WED-state to a state correspondent to a final

WED-state in its equivalent TPN.

In the reachability tree of a TPN, the root node represents the initial state, and each child node recursively represents a state reachable from the state represented by its parent node through the firing of a single transition or the elapse of time [PZ13]. Adding up the elapsed times in the path from the root node to each other node in this reachability tree thus gives the time taken for the initial state to reach that other state.

Now consider a Time p-WED-flow process instance. In order to calculate the minimum distance of time from a state correspondent to its current WED-state to a state correspondent to the final WED-state, transitions fire when they become time-enabled. Also, if no transition is time-enabled for a certain state, the minimum required time for some transition to become time-enabled is elapsed. Analogously, in order to calculate the maximum distance of time from a state correspondent to its current WED-state to a state correspondent to the final WED-state, transitions fire as late as they can, and if no transition is time-enabled for a certain state, the maximum possible time is elapsed.

Chapter 4

SISAUT

Autopsies determine the cause of death for a death certificate, but the deceased may have chosen (and their families may have consented) to donate their bodies for scientific research. On average, the University of Sao Paulo (USP) performs 40 autopsies per day as a public service, of which some lead to the collection and processing of organs for scientific research. In short, for organs to be used in research projects, two interacting, parallel, time-critical processes have to be successfully completed: (1) the collection and processing of materials, and (2) the filling of consent forms and questionnaires.

Both processes are initialized when a deceased body is declared of scientific interest. First, personal and demographic data are gathered in parallel with the register of body measurements. Suitable research projects are then chosen considering the information collected so far. Next, appropriate consent forms have to be filled out by family members, and consented research projects have to be selected (possibly resolving conflicts among consented research projects). At this point, the set of organs to be collected and the processing techniques to be performed have been already defined, and the collection of materials is authorized. Family members are also expected to fill out specific questionnaires related to those selected research projects. After materials are collected, they can be processed, but first their conditions have to be checked. Furthermore, materials can only be used in research projects if their processing is completed within 24 hours from the declared time of death.

In this Chapter, we model SISAUT – the autopsy management system that coordinates these two processes – using Time WED-flow. In particular, the SISAUT process model is a Time p-WED-flow process model. We underline that we only model the critical path of those processes, although other execution paths could exist (e.g., the family did not consent), because only the critical path

is of interest of a timeliness check.

Based on the previous description, the WED-transitions encapsulating the execution of tasks of these processes are the following:

1. *gather_data*;
2. *choose_projects*;
3. *family_consent*;
4. *select_consented_projects*;
5. *fill_out_questionnaires*;
6. *register_body_measurements*;
7. *collect_materials*;
8. *check_materials*;
9. *process_materials*.

For the sake of simplicity, we refer to each of these WED-transitions as t_k, $1 \leqslant k \leqslant 9$, where k is the number of the WED-transition in the enumeration above. For example, *select_consented_projects* is referred to as t_4. As a p-WED-flow process model, each of these WED-transitions updates the value of a correspondent WED-attribute to *true*. Here we adopt the convention that the WED-attribute whose value is updated to *true* by the WED-transition t_k is referred to as a_k.

The WED-triggers of the interview process are the following:

- $(a_1 = false, t_1)$;
- $(a_1 = true$ and $a_2 = false, t_2)$;
- $(a_2 = true$ and $a_3 = false, t_3)$;
- $(a_3 = true$ and $a_4 = false, t_4)$;
- $(a_4 = true$ and $a_5 = false, t_5)$.

The WED-triggers of the process for collecting materials are the following:

- $(a_6 = false, t_6)$;
- $(a_4 = true$ and $a_6 = true$ and $a_7 = false, t_7)$.

The WED-triggers of the process for processing materials are the following:

- $(a_7 = true \text{ and } a_8 = false, t_8)$;
- $(a_8 = true \text{ and } a_9 = false, t_9)$.

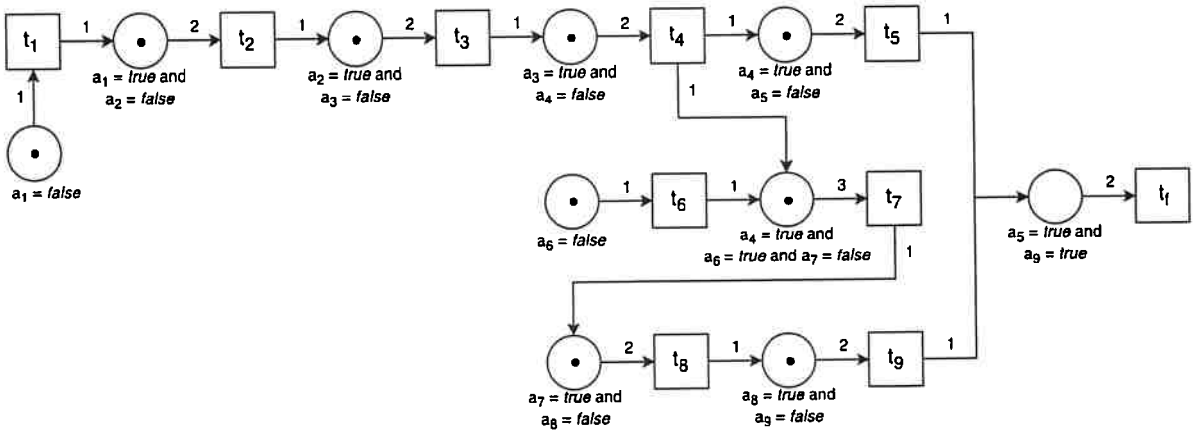


Figure 4.1: Initial marking of the Petri net equivalent to the p-WED-flow process model of SISAUT.

```
create wed-instance as default values
        with timeout time_of_death - interval '24 hours';
```

Listing 4.1: Instantiating the SISAUT WED-flow process model.

Since we specify a final deadline for SISAUT instances in terms of a time WED-attribute *time_of_death*, we remark that a p-WED-flow might have other non-boolean WED-attributes. These WED-attributes, however, cannot be evaluated in WED-conditions.

In the following table, we assign minimum and maximum execution times in minutes to each WED-transition of the p-WED-flow process model of SISAUT. Such minimum and maximum execution times are the best- and worst-case estimates of the time it takes to perform the task correspondent to that WED-transition, respectively. We remark that these execution times are relative to the time a WED-transition is triggered. Furthermore, they are fictitious.

WED-transition	Min. exec. time (min.)	Max. exec. time (min.)
gather_data (t_1)	30	120
choose_projects (t_2)	30	120
family_consent (t_3)	30	1080
select_consented_projects (t_4)	30	120
fill_out_questionnaires (t_5)	30	180
register_body_measurements (t_6)	30	360
collect_materials (t_7)	60	120
check_materials (t_8)	30	60
process_materials (t_9)	30	120

For some cases, consented projects might have been selected and body measurements might have been registered, but materials might have not been collected. For others, only the processing phase might be missing. Since each case has its own final deadline dependent on the time of death, and the number of health professionals available is limited, it becomes essential to assign a priority to each case in order to avoid spending resources for collecting and processing materials of cases that are unlikely to meet their deadlines. Therefore, we calculate both best- and worst-case estimates of the time it takes to perform the remaining tasks of each case. A low priority could then be assigned to a case which is not expected to meet its final deadline by the best-case estimate. Also, a set of cases which are expected to meet their deadlines by the best-case estimate but not by the worst-case estimate could be prioritized according to how far their worst-case estimates are from their final deadlines.

Chapter 5

Conclusion

In this work, we have extended the WED-flow process modeling approach with the notion of time. The resulting Time WED-flow model is suitable for modeling time-critical processes usually found in real-time systems. We also have presented a method for mapping a p-WED-flow process model to a Petri net. Then, the control-flow logic of p-WED-flow process models can be checked using a standard reachability analysis. We also presented the Time Petri net model for describing the temporal semantics of Time p-WED-flow process models. Once more, standard reachability analysis techniques can be used, but now for the timeliness check of those Time p-WED-flow process models. Also, the practical aspects of the implementation of Time WED-flows were focused on WED-SQL, aiming to provide a foundation for its implementation of the time perspective of processes.

A direction for further research is to model the temporal semantics of Time WED-flow using a stochastic Petri net model. In this way, we would be able to not only evaluate best- and worst-case scenarios. Other direction is to model more classes of WED-flows using colored Petri nets, which increase the expressiveness of the Petri net model, while increasing the cost for checking most of its interesting properties.

Bibliography

[ACD90] Rajeev Alur, Costas Courcoubetis e David Dill. Model-checking for real-time systems. Em *Symposium on Logic in Computer Science*, páginas 414–425. IEEE, 1990. 3

[Att04] Mohsen Attaran. Exploring the relationship between information technology and business process reengineering. *Information & Management*, 41(5):585–596, 2004. 2

[BJ94] Christoph Bussler e Stefan Jablonski. Implementing agent coordination for workflow management systems using active database systems. Em *International Workshop on Research Issues in Data Engineering*, páginas 53–59. IEEE, 1994. 2

[CGP99] Edmund M. Clarke, Orna Grumberg e Doron Peled. *Model checking*. MIT press, 1999. 3

[CH09] David Cohn e Richard Hull. Business artifacts: a data-centric approach to modeling business operations and processes. *IEEE Data Engineering Bulletin*, 32(3):3–9, 2009. 2

[DvdAtH] Marlon Dumas, Wil M. P. van der Aalst e Arthur H. M. ter Hosfetde. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons. 1

[FBTP12] Joao E. Ferreira, Kelly R. Braghetto, Osvaldo K. Takai e Calton Pu. Transactional recovery support for robust exception handling in business process services. Em *International Conference on Web Services*, páginas 303–310. IEEE, 2012. 2, 5, 14

[FTK+] Joao E. Ferreira, Pedro L. Takecian, Leonardo T. Kamamura, Bruno Padilha e Calton Pu. Dependency management with wed-flow techniques and tools: a case study. Em *International Conference on Collaboration and Internet Computing*. IEEE. 2

[GMS87] Hector Garcia-Molina e Kenneth Salem. Sagas. Em *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, páginas 249–259. ACM, 1987. 5

[GR92] Jim Gray e Andreas Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992. 2, 5, 6

[HM96] Constance Heitmeyer e Dino Mandrioli. *Formal methods for real-time computing*. John Wiley & Sons, Inc., 1996. 3

[HM10] Thomas T. Hildebrandt e Raghava R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. Em *PLACES 2010*, páginas 59–73, 2010. 2

[Hol97] Gerard J. Holzmann. The model checker spin. *Transactions on software engineering*, 23(5):279–295, 1997. 3

[KRSR96] Gerti Kappel, Stefan Rausch-Schott e Werner Retschitzegger. Coordination in workflow management systems – a rule-based approach. Em *Annual Asian Computing Science Conference*, páginas 99–119. Springer, 1996. 2

[Lam02] Leslie Lamport. *Specifying systems: the TLA+ language and tools for hardware and software engineers.* Addison-Wesley, 2002. 3

[Lap04] Phillip A. Laplante. *Real-time systems design and analysis.* Wiley New York, 2004. 3

[LS07] Ruopeng Lu e Shazia Sadiq. A survey of comparative business process modeling approaches. Em *Business information systems*, páginas 82–94. Springer, 2007. 2

[Mur89] Tadao Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. 3, 7

[Pet62] Carl A. Petri. *Kommunikation mit automaten.* Tese de Doutorado, Universitat Hamburg, 1962. 7

[pos17] Postgresql database management system, 2017. 13

[PSR17] Bruno Padilha, Andre L. Schwerz e Rafael L. Roberto. Wed-sql: A relational framework for design and implementation of process-aware information systems. Em *International Conference on Distributed Computing Systems Workshops*, páginas 364–369. IEEE, 2017. 13

[PVdA06] Maja Pesic e Wil M. P. Van der Aalst. A declarative approach for flexible business processes management. Em *International conference on business process management*, páginas 169–180. Springer, 2006. 2

[PZ13] Louchka Popova-Zeugmann. *Time and Petri Nets.* Springer, 2013. 16, 18

[RW12] Manfred Reichert e Barbara Weber. *Enabling flexibility in process-aware information systems: challenges, methods, technologies.* Springer Science & Business Media, 2012. 1

[VDATH05] Wil M. P. Van Der Aalst e Arthur H. M. Ter Hofstede. Yawl: yet another workflow language. *Information systems*, 30(4):245–275, 2005. 2

[VdAWG05] Wil M. P. Van der Aalst, Mathias Weske e Dolf Grunbauer. Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, 53(2):129–162, 2005. 2

[Wan98] Jiacun Wang. *Timed Petri Nets.* Springer, 1998. 15

[Whi04] Stephen A. White. Introduction to bpmn. 2004. 2