MASTER

Mining simulation models with correlations

Jian, J.

Award date:
2009

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MINING SIMULATION MODELS WITH CORRELATIONS

By
Jingxian Jian

SUPERVISED BY
dr.N.Sidorova
ir.M.H.Schonenberg

Table of Contents

Acknowledgements

First and foremost, I would like to thank my academic supervisors, Natalia Sidorova and Helen Schonenberg, for being excellent supervisors in the past 6 months. Their excellent supervision made it possible for me to accomplish this thesis.

I am also grateful to Carmen Bratosin and Eric Verbeek who gave me valuable suggestions related to the ProM framework. I would also like to thank Toon Calders for the presence of my presentation, giving me valuable feedback.

Special thanks to my lovely boyfriend, Grant Patrizio Kesuma, who always supports me all the time. Because of him, my life is so meaningful in Netherlands. I would like to thank my uncle and auntie for taking care of my lovely mother when I was not beside her. At last, but the most important, I would like to thank my father and my mother for their continuous support and encouragement. They have sacrificed so much for supporting me to pursue my master degree abroad. I would like to dedicate this thesis to both of them.

Eindhoven, Noord Brabant Jingxian Jian
August 02, 2009

Abstract

Computer simulation is a useful method for analyzing information systems as it enables "what if" analysis, i.e. looking into the future under certain assumptions [20]. In this thesis, simulation models are used to estimate the eventual effect of changes in the process. Simulation models are supposed to represent certain key characteristics or behaviors of the business processes. One of these key behaviors is the decision making in the process.

A traditional way to obtain such simulation models is to create them manually, according to documentation and close observation of the real systems. However, it is rather error-prone and time-consuming. In [15], Rozinat et al. introduced a methodology to derive simulation models with data dependencies or probability distributions from logs, which are historical executions recorded by information systems. Real business processes involve decisions, which are possibly correlated globally, i.e. one decision may be affected by the decisions made before. As data is not always available in logs, simulation models with data dependencies can not be mined in most cases. Moreover, data dependencies in real systems are often much more complex than the ones that can be mined with the existing techniques. For derived simulation models [15] with probability distributions, the correlations between the decisions and the change propagation capability are missing.

A simulation modeling framework, called History-Dependent Stochastic Petri Nets (HDSPNs) [16], was proposed to derive simulation models from logs including history-dependent correlations. The expectation was that this framework will allow to propagate process changes by taking the correlations and history into account.

The goal of this thesis is to verify whether HDSPNs are able to propagate process changes through the process, especially on the decisions. To archive so, process mining algorithms are introduced to discover correlations. Based on an example process which produces logs, the HDSPN is constructed with the correlations mined from the log which is generated by the example process. Before introducing process changes, we validate whether the HDSPN approximates the behavior of the example process well enough by comparing the frequencies of certain decisions in the process. To verify the propagation of changes, process changes are injected into the example process and the HDSPN. The frequencies of the decisions in the example process and the HDSPN are measured after the change injection.

The simulation results show to us that when process changes are injected, HDSPNs are able to propagate process changes in the process. The quality of the approximation obtained for HDSPNs depends on the relationship between process changes introduced, the correlations incorporated and the abstractions used for mining.

Introduction

Computer simulation is a useful method for analyzing information systems as it enables precisely “what if” analysis, i.e. to look into the future under certain assumptions [20]. Simulation models are needed to drive the simulation experiments. In this thesis, they are used to estimate the eventual effect of changes in the process.

Simulation models are supposed to represent certain key characteristics or behaviors of business processes. One of these key behaviors is the decision making in the processes. Normally, business processes involve decisions which are possibly correlated globally, i.e. one decision may be affected by the decisions made before. Once some process change emerges, e.g. economic factors or client profiles, due to the existence of correlations, the change is propagated through the process. For instance, consider a travel agency process. The decision of the client to book a pickup service is mainly influenced by the distance of the hotel booked from the airport and by the client’s budget. So we can expect the existence of a correlation between the pickup booking decision and the hotel choice. When some process changes emerge, e.g. hotels near the airport are more preferred, probably fewer pickup services would be booked, i.e. the change will be probably propagated to the correlated decision. Therefore, simulation models are required to take correlations into account and be able to propagate process changes.

A traditional way to obtain such simulation models is to create them manually, according to documentation and close observation of the real systems. However, it is rather error-prone and time-consuming. In [15], Rozinat et al. introduced a methodology to derive simulation models including data dependencies or probability distributions from logs, which are historical executions recorded by information systems. Data dependencies attach data attributes to decisions. For instance, the decision whether to book a pickup service is mainly influenced by the distance from the hotel booked to the airport. The distance here can be a data attribute binding two decisions. If the client decides to change the hotel, the corresponding distance is changed and the pickup booking decision is probably changed as well. Simulation models with data dependencies indeed address process changes locally, i.e. where the data attributes are used. However, in the reality, logs do not always record these data attributes. Moreover, data dependencies in real systems are often much more complex than the ones that can be mined with the existing techniques. Therefore, simulation models including data dependencies are not always derivable. Another option Rozinat et al suggested is to incorporate frequencies in the simulation models. By using probability distributions derived from the history, they proposed to replace simulation models with complex data dependencies by Petri nets with frequencies. However, Petri nets with frequencies fail to propagate process changes due to the absence of the correlations between decisions.

In [16], a simulation modeling framework called History-Dependent Stochastic Petri Nets (HDSPNs) was introduced. HDSPN is an extension of a classical Petri net with probability distributions which considers not only the history but also the correlations. HDSPNs are assumed to be able to propagate the process changes well by taking correlations into account. However, the existing HDSPNs framework [16] introduced a general approach to discover correlations among decisions and was not applied to a case study.

In this thesis, our goal is to assess the hypothesis that HDSPNs are able to propagate process changes through the process. The methodology shown in Figure 1 to assert our hypothesis is summarized as follows:

1. Acquire a log and the corresponding control flow of an example process to be used in next steps;

2. Introduce algorithms to discover correlations and implement them as a plug-in in the ProM framework;

3. Develop a HDSPN with mined correlations in CPN tools;

4. Validation: assess if the HDSPN approximates the behavior of the example process well enough before the introduction of process changes;

5. Change propagation analysis: assess if the HDSPN propagates the process changes, i.e. whether it approximates the behavior of the example process well enough after the introduction of process changes.

First of all, we need a log and the corresponding control flow from a concrete business process. For our research, we are interested in the propagation of process changes through the process, which is not in any existing logs yet. To observe the change propagation, process changes need to be introduced to the process. It is too risky and costly to ask a company or an organization to execute their processes with respect to the process changes. Consequently, we construct an example process which contains some correlations between decisions (introduced in Chapter 4).

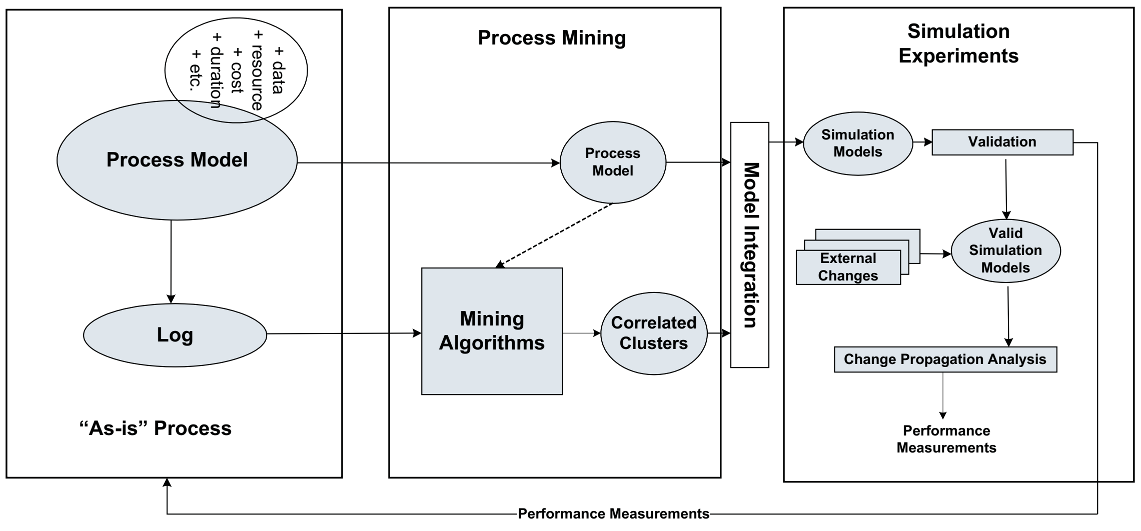


Figure 1: Overview of our methodology.

To discover correlations between decisions from the log generated by the example process, specific process mining algorithms are introduced and applied (introduced in Chapter 3). All ingredients, e.g. the correlations, the history, and the control flow, are then integrated into our simulation model, i.e. HDSPN (introduced in Chapter 2).

To make sure that the simulation model is a reasonable representation of the example process, the behaviors of simulation models without process changes are then validated. Without applying process changes to a decision in the process, we measure the frequencies of the correlated decisions for both the HDSPN and the example process. Comparison is conducted based on the simulation measurements.

For the valid HDSPN, which produces similar frequencies of decisions as the example process does, change propagation analysis is conducted. Change propagation analysis assesses if the HDSPN propagates the process changes well. With applying process changes to certain decisions in the process, we again measure the frequencies of the correlated decisions for both HDSPN and the example process. Comparison is conducted based on the simulation measurements. Detailed experimental settings and results of the validation and the change propagation analysis are discussed in Chapter 4 and Chapter 5 respectively. Conclusions and future work are discussed in Chapter 6.

Chapter 1

Preliminaries

In this chapter, we introduce some basic concepts.

$\mathbb{N}$ denotes the set of natural numbers. Let S be a set. A *multiset* $\mathcal{M}$ over S, i.e. $\mathcal{M}(S)$ is a mapping $\mathcal{M} : S \rightarrow \mathbb{N}$, with $domain(\mathcal{M}) = S$. Suppose $S = \{a, b, c\}$; an example of a *multiset* over S could be $\{a^2, b, c^3\}$, which is a multiset containing two a's, one b, and three c's. We denote the sum of *multisets*, the difference of *multisets*, and *sub-multisets* by $+$, $-$, and $\leq$ respectively. The notations $|S|$ and $|\mathcal{M}|$ are used to denote the number of the elements in the set and *multiset* respectively.

Let S be a set. A sequence over S of length n is a function $\sigma : \{0, \ldots, n-1\} \rightarrow S$. It is represented by a string, e.g. $\sigma = \langle a_0, \ldots, a_{n-1} \rangle$ where $\sigma(i) = a_i$ for $i \in [0, n-1]$. The length of a sequence is denoted by $|\sigma|$. The sequence of length 0 is called the empty sequence and is denoted by ε. The set of infinite sequences over S is denoted by S^*. The concatenation $\sigma; s$ of sequence $\sigma = \langle a_0, \ldots, a_{n-1} \rangle$ with $s \in S$ is the sequence $\langle a_0, \ldots, a_{n-1}, s \rangle$, and the concatenation $\sigma; \gamma$ of σ with sequence $\gamma = \langle b_0, \ldots, b_{n-1} \rangle$ is the sequence $\langle a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1} \rangle$.

Let S be a set and σ be a sequence. A *projection* $\sigma \upharpoonright S$ is the *projection* of σ over a set S, removing all elements from σ that are not in S. For instance, $S = \{a, b, c\}$ and $\sigma = \langle a, b, b, b, c, d, c, e \rangle$, then $\sigma \upharpoonright S = \langle a, b, b, b, c, d, c, e \rangle \upharpoonright \{a, b, c\} = \langle a, b, b, b, c, c \rangle$.

For any sequence σ over S, the Parikh vector $par(\sigma)$ maps every element s of S onto the number of occurrences of s in σ, i.e. $par(\sigma) \in \mathcal{M}(S)$ where for any $s \in S : par(\sigma)(s) = |\sigma \upharpoonright \{s\}|$. Suppose $S = \{a, b, c\}$ and $\sigma = \langle a, b, b, b, c, d, c, e \rangle$, then $par(\sigma) = \{a^1, b^3, c^2\}$. For any $s \in S$, $par(\sigma)(s)$ counts the frequency of s in σ.

1.1 Business Process

Business processes consist of a collection of related *tasks* that produce a specific service or product in a structured order [19]. A *task* here refers to a logical unit of work, e.g. registering an application, sending an email, filling a complaint. This definition implies that *business processes* are *case-based*, which means that every piece of work is executed for a specific *case*. A concrete example of a *case* is a tax declaration, an order, or a request for information. Let us take a running example shown in Figure 1.1 as an example to illustrate these definitions.

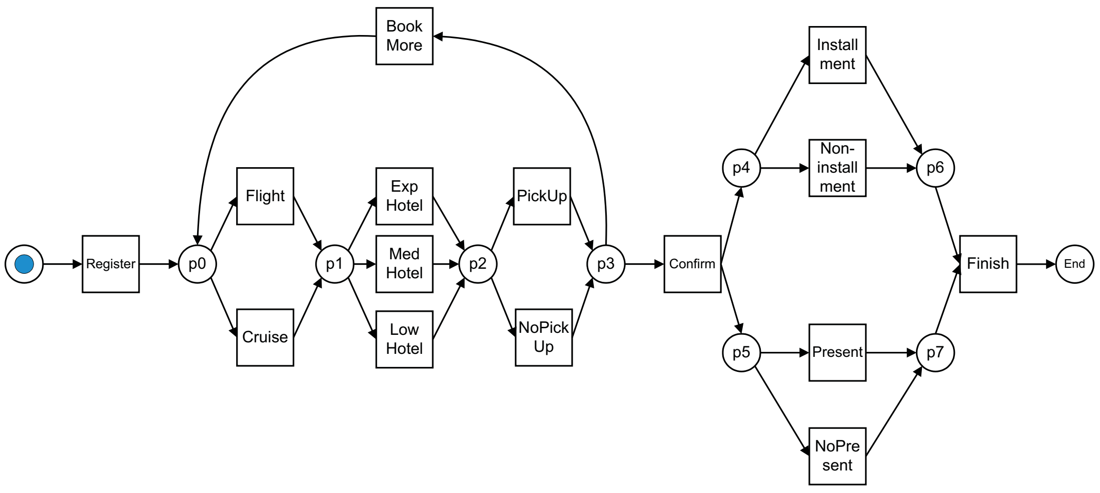


Figure 1.1: Example process: travel agency process.

Figure 1.1 outlines the processing of a travel package booking request within a travel agency. The process starts with the registration (*Register* in Figure 1.1). Some data related to the request is registered. Then either flight or cruise for transportation is chosen (*Flight* or *Cruise*). The requirement for a pick-up service (*PickUp* or *NoPickUp*) can be put after a hotel is booked. Three classes of hotel are available, i.e. luxury (*ExpHotel*), middle class (*MedHotel*), and budget hotels (*LowHotel*). Afterwards, either confirmation or going back to book more packages is submitted. Once confirmation is submitted, the payment type is checked (*Installment* or *Non-installment*) and the present option for special clients (*Present* or *NoPresent*) is evaluated. Finally, the case is archived and closed (*Finish*).

In this example, we define the procedures how the agency handles customers' requests as the *business process* where a customer's request is a *case.* Units of work such as "register", "book flights", and "book expensive hotels" are concrete examples of *tasks.* The order in which *tasks* are performed is called *sequence.* A *task* here is a generic piece of work rather than an actual one. It becomes an actual piece of work only when it is bound with a specific *case.* The term *activity* refers to the execution of an actual piece of work. It means that as soon as a *task* combined with a *case* is executed, it is considered to be an *activity.*

1.2 Petri Nets

Petri nets is one of the formal modeling languages used to describe the structure of tasks in information systems. A Petri net, consisting of nodes (places and transitions) and arcs, represents logical interactions among activities in a natural way. Places in Petri nets identify the processing status of the system (working, idle, queuing, failed, etc.), and transitions describe the passage from one status to another (end of a task, failure, repair, etc.) [5]. In the context of Petri nets, a *business process* refers to a *net*, a *task* to a *transition*, and a *case* to an instance of the *net.*

Typical system properties, such as synchronization, sequentiality, iteration, and parallelism, can be modeled by Petri nets. Besides, abundant properties have been formally defined, including liveness, boundedness, safety, as well as analysis techniques of Petri nets, e.g. reachability graph and reachability trees [14]. These formally defined properties facilitate the analysis of information systems. Thus, due to the formal semantics of Petri nets, explicit states modeling, and the abundance of theoretically proven analysis techniques, gradually, the use of Petri net-based models has become a widely used practice for modeling information systems in academia [17].

Definition 1.1 (Petri nets). A Petri net is defined as a tuple $\langle P, T, F\rangle$ where:

1. P is a finite set of places,
2. T is a finite set of transitions such that $P \cap T = \emptyset$ and
3. $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a flow relation mapping pairs of places and transitions to the naturals such that for any pair of nodes (x, y) with $F(x, y) \geq 1$, $F(x, y)$ is the weight of an arc (x, y) .

Given $t \in T$, the preset ${}^\bullet t$ and the postset $t^\bullet$ of t are the multisets of places where every $p \in P$ occurs $F(p, t)$ times in ${}^\bullet t$ and $F(t, p)$ times in $t^\bullet$.

A marking m of a net $N = \langle P, T, F\rangle$ is a multiset over P; markings are states of a net. A pair (N, m) is called a marked Petri net. A transition $t \in T$ is enabled in marking m if and only if $^\bullet t \leq m$. An enabled transition t may fire. This results in a new marking m' defined by $m' = m -^\bullet t + t^\bullet$.

The process in Figure 1.1 is modeled as a Petri net. Suppose there is one token in place p_0, i.e. the marking is $\{p_0^1\}$. Since $^\bullet Flight$ and $^\bullet Cruise$ both are $\{p_0\}$, and $\{p_0\}$ is a sub-multiset of $\{p_0^1\}$, both transitions, $Flight$ and $Cruise$, are *enabled*, although only one of them can *fire*. No matter which transition fires, one token is consumed from the input place and produced to the output place. The resulting marking will be $\{p_1^1\}$. Now, $Flight$ and $Cruise$ are no longer *enabled*.

A Petri net class of particular practical interests for us is free-choice Petri nets [7]. In these nets, choice and synchonisation are separated like in many other graphical process modeling notations. Moreover, free-choice nets are derived by many process mining algorithms [18].

Definition 1.2 (**Free–choice nets**)**.** A net, $N = \langle P, T, F \rangle$ is a free–choice if for every transition t_1 and t_2, $^\bullet t_1 \cap^\bullet t_2 \neq \emptyset$ implies that $^\bullet t_1 =^\bullet t_2$.

In [7], the following important property of free-choice nets is introduced. When a transition t is enabled in a marking m, all other transitions in t's *Tcluster* are enabled as well.

Definition 1.3 (**TCluster**)**.** Let $t \in T$ of a Petri net $N = \langle P, T, F \rangle$. The cluster of t, denoted by $[t]$, is the set of transitions such that

- $t \in [t]$,
- For every $t' \in T$, $t' \in [t]$ iff $^\bullet t' =^\bullet t$.

Note that such *TCluster* is different from the traditional concept of *cluster* in [7], which contains both places and transitions. In the remainder of this thesis, all "*clusters*" or notations $[t]$ refer to *TClusters* only. Concrete examples of *TClusters* in Figure 1.1 are $[Flight] = [Cruise] = \{Flight, Cruise\}$, $[Register] = \{Register\}$, $[ExpHotel] = [MedHotel] = [LowHotel] = \{ExpHotel, MedHotel, LowHotel\}$, etc.

Definition 1.4 (**Cluster set**)**.** Let $t \in T$ of a Petri net $N = \langle P, T, F \rangle$. The cluster set denoted by C is the set of $TClusters$ of N, i.e. $C = \{[x] | x \in T\}$.

1.3 PAISs and Logs

Currently, many companies have adopted information systems that are configured on the basis of process models to support their business processes in some form. Such information systems are so-called Process Aware Information Systems (PAISs) [12]. In such systems, explicit process models describe how the business processes should be executed. In addition, these systems typically log *events* (e.g. in transaction logs or audit trails) related to the actual business process executions. An *event* is the reflection in the log of an *activity* in the real life. The concept of *event* is defined as follows.

Definition 1.5 (**Event**). Let E be a set of *events.* Every $e \in E$ is a touple consisting of the event name, case ID and possibly some additional information.

An *event* logged in most information systems includes not only the name of the execution and the case ID but also other additional properties, such as timestamp and the originator. All these properties can be used for different process discovery techniques. However, in this thesis, only *case ID* is taken into account, as other additional properties are often absent.

Definition 1.6 (**Trace**). We define a *trace* λ as a sequence over *event* names and case IDs.

Referring to the case IDs, traces having same case ID are grouped. Without the timestamps, originators, or other properties in events, one concrete example of a *trace* for the process in Figure 1.1 could be $\langle Register, Flight, ExpHotel, PickUp, Confirm, Installment,$ $Present, Finish\rangle$ corresponding to case ID 1. A log contains finite traces.

Definition 1.7 (**Log**). A *log* L is a set of traces.

1.4 Markov Processes

A function $f : A \times B \to [0, 1]$, where A and B are finite or countable sets, is called a transition probability function if for all $b \in B : f(., b)$ is a probability over A, i.e. for all $b \in B : \sum_{a \in A} f(a, b) = 1$.

A discrete stochastic process is a finite or infinite sequence of random variable $X_1, X_2, X_3, \ldots$ with values in some domain X, defined on some probability space $(\Omega, \Im, \mathbb{P})$, where Ω is the sample space, $\Im$ is a σ-algebra on Ω and $\mathbb{P}$ is a probability measure on $\Im$, such that $\mathbb{P}(\emptyset) = 1 - \mathbb{P}(\Omega) = 0$. We characterize $\mathbb{P}$ without explicit construction of Ω and $\Im$, by conditional probabilities:

$$\mathbb{P}[X_{n+1} = y | X_0 = x_0, \ldots, X_n = x_n] = f(y, \langle x_0, \ldots, x_n \rangle)$$

for $y \in X$ and $x_i \in X$, $i = 0, 1, 2, \ldots, n$. We assume that transition probabitlity f is a computable function on $X \times X^*$. Note that by the theory of Ionescu Tulcea (see [11]) the transition probability function characterizes the probability measure $\mathbb{P}$ completely. In fact, this is a generalization of the result of A. Kolmogorove presented in [4].

Chapter 2

Models for Simulation Experiments

In this thesis, we intend to estimate the eventual effect of process changes through the process by means of computer simulations. Computer simulation is a useful method for analyzing information systems as it enables “what if” analysis, i.e. looking into the future under certain assumptions [20]. In our case, we investigate *what* the frequencies of the decisions in the process will be *if* process changes are injected into the process. Let us take the process in Figure 1.1 as an example. In the travel agency process, there is a pickup booking decision and a process change that more clients book hotels near the airport. “What if” analysis helps predict the average number of pickup services booked when more such hotels are reserved. This kind of prediction is unlikely to obtain by experiments running in real systems.

Normally simulations are driven by models. To simulate the eventual effects of certain alternatives in a computer, simulation models are required to represent certain key behaviors of the actual systems [20]. One of these key behaviors in real systems is the decision making. Normally, business processes involve decisions which are possibly correlated globally, i.e. one decision may be affected by the decisions made before. Once some process change emerges on some decision, e.g. economic factors or client profiles, due to the existence of correlations, the correlated decisions should be affected, i.e. change is propagated through the process. For instance, consider a travel agency process. The decision of the client to book a pickup service is mainly influenced by the distance of the hotel booked from the airport and by the client’s budget. So we can expect the existence of a correlation between the pickup booking decision and the hotel choice. When some process changes emerge, e.g. hotels near the airport are more preferred, probably fewer pickup services would be booked, i.e. the change will be probably propagated to the correlated decision. Therefore, for simulating the change propagation, the simulation model is required to take correlations into account and be able to propagate process changes.

Traditionally, simulation models are generated manually according to human being’s understandings of the real processes after a series of interviews, close observation, and documentation studies have been done. However, this method is rather time-consuming and possibly error-prone. In [15], Rozinat et al. introduced a methodology to discover the simulation models from logs. To mimic the decisions made in the real systems, they suggested two alternatives: 1) to discover and involve data dependencies; and 2) to discover and involve probability distributions in simulation models.

For alternative 1), data dependencies attach data attributes to decisions. For instance, a decision to book a five-star hotel or a hostel is dependent on the budget. Budget here is the data attribute. If the budget changes, the decision on the hotel (or hostel) could change as well. Simulation models with data dependencies indeed address process changes locally, i.e. where the data attributes are used. However, in the reality, logs do not always record these data attributes. Moreover, data dependencies in real systems are often much more complex than the ones that can be mined with the existing techniques. Therefore, simulation models including data dependencies are not always derivable from logs. For alternative 2), simulation models with complex dependencies are replaced by Petri nets with frequencies. A firing probability distribution, derived from the log, is assigned to each alternative of the choices to mimic the decision making by users. For instance, a decision to book a five-star hotel or a hostel is dependent on the budget. However, the data attribute about budget is not recorded in the log. It is only known from the log that 70% of the clients who booked five-star hotels while 30% for hostels. Instead of the data dependencies, it is proposed to assign 70% to the alternative of the five-star hotel and 30% to the hostel. According to the experimental results, Petri nets with frequencies approximate the behavior of the real systems well with respect to a number of performance indicators. However, they are not able to propagate changes through the process due to the absence of the correlations. Thus, Petri nets with frequencies are unsuitable for simulating the change propagation.

A simulation modeling framework called History-Dependent Stochastic Petri nets (HDSPNs) [16] has caught our eyes as it considers not only the history but also the correlations. To build HDSPNs, additional elements are needed to be incorporated for simulation purpose. Essential elements include (1) the history-dependent transition probabilities extracted from logs; and (2) the global history. In the remainder of this chapter, details about HDSPNs are introduced.

2.1 History-Dependent Stochastic Petri Nets

In this section, we give a definition of our simulation model, History-Dependent Stochastic Petri Nets, as well as definitions of history-dependent probability measures.

A History-Dependent Stochastic Petri net is an extension of a classical Petri net with history [22] denoted by H, dummy transition δ, and transition probability function denoted by f.

Definition 2.1 (HDSPNs). A History-Dependent Stochastic Petri net (HDSPN) N is a tuple $\langle P, T, F, m_0, f\rangle$, where $N = \langle P, T \cup \{\delta\}, F\rangle$ is a Petri net with $^\bullet\delta = \delta^\bullet = \emptyset$, m_0 is an initial marking, and $f : (T \cup \{\delta\}) \times H \to [0, 1]$ is a transition probability function, depending on the history H.

From the definition, it is explicit that an HDSPN is indeed an extension of a classical Petri net. Moreover, HDSPN exhibits the behavior of a stochastic process which is described as a sequence of random variables $X_1, X_2, X_3, \ldots$, where the domain of X is $X \cup \{\delta\}$, X_n denotes the corresponding n^{th} transition of a marked Petri net (N, m_0) that fires from an initial marking m_0. δ is an isolated dummy transition and enables nets (business processes) to keeping firing infinitely even when a deadlock occurs or the final state is reached. For convenience, shorthand notation H_n denotes $X_1, \ldots, X_n$, the domain of H is X^*, i.e. the set of all possible histories. Note that the transition probability function f in the definition is vital for the generation of history-dependent transition probabilities. The formal definition of f is as follows.

Definition 2.2 (Transition Probability Function $f(t, h)$). For $t \in T$, $h \in H$, we define $\mathbb{P}[X_n = t|H_n = h] = f(t, h)$, where $f : T \times H \to [0, 1]$ is the transition probability function. $f(t, h)$ is assumed to satisfy: (1) the sum of the probabilities over all transitions and dummy transition δ is 1; (2) the transition probability function of a transition equals to 0 if this transition is disabled; (3) δ can fire if and only if every transition in the net is disabled.

To generate history-dependent transition probabilities for free choice Petri nets, the approach from [16] to further specify $f(t, h)$ is applied. For free-choice Petri nets, when a transition in a cluster is enabled, all transitions in this cluster are enabled. The selection of a transition to fire can be seen as a procedure first to choose an enabled cluster and then a transition from this cluster to fire. As a result, the transition probability function $f(t, h)$ can be further specified as

$$f(t, h) = p(t, h) \cdot q(t, h),$$

where $p(t, h)$ is the probability to select transition t in a cluster after history h and $q(t, h)$ is the probability to select cluster $[t]$.

As choosing one cluster to fire from several simultaneously enabled clusters will not disable other enabled clusters, we assume that the probability to choose one cluster from a population of simultaneously enabled clusters is equal, i.e. q(t, h) for every simultaneously enabled clusters is same. Then, the problem of determining $f(t, h)$ is restricted to determining $p(t, h)$. In real life, $p(t, h)$ is usually unknown. We propose to estimate $p(t, h)$ on the logs which record history. The formal definition of $p(t, h)$ is given in Section 2.4.

2.2 Abstractions

As mentioned previously, we can estimate $p(t, h)$ by using the logs. A naive way to approximate $p(t, h)$ is to count the number of the occurrences of the transition t after whole history h in the log. However, if we consider the entire history h for estimating the transition probability for a transition t, we will normally find out that we have not enough data in our log to make any reliable estimations.

Let us take a concrete example to illustrate the motivation to introduce abstractions. In the travel agency example process in Figure 1.1, we suppose that one case is running in the process and we have a log L. When the decision $PickUp$ or $NoPickUp$ is going to proceed for the first time, the observed history $h = \langle Flight, ExpHotel \rangle$, which never happened in the log before. Since h is never observed in L, no reliable estimation of the transition probability can be estimated on the log. However, if we only consider part of history h, i.e. $h' = \langle ExpHotel \rangle$, and h' is observed in L, a reliable estimation of the transition probability can be obtained from the log. It shows that many possible situations would be omitted when using the whole history to estimate $p(t, h)$. Therefore, abstractions are preferred to be used for the history h. Here we introduce two types of abstractions, *transition frequency abstraction* and *last fired cluster transition abstraction*.

Definition 2.3 (Transition Frequency (TF) Abstraction). Let C be a cluster set of a net $\langle P, T, F \rangle$, $c \in C$, and H be the *history*. We define the *transition frequency abstraction* $\alpha^{tf} : H \times C \to par(H \uparrow C)$:

$$\alpha^{tf}(h, c) = par(h \uparrow c),$$

where $par(h \uparrow c)$ projects history h on cluster c and results in a multiset over c.

The transition frequency abstraction actually observes the frequencies of all transitions from the defined cluster in the history. For instance, let $h = \langle a, x, y, b, a, a, b \rangle$, $c_1 = \{a, b, c\}$, and $c_2 = \{e, f\}$, then $\alpha^{tf}(h, c_1) = \{a^3, b^2, c^0\}$ and $\alpha^{tf}(h, c_2) = \{e^0, f^0\}$. If $h = \varepsilon$, then $\alpha^{tf}(h, c_1) = \{a^0, b^0, c^0\}$ and $\alpha^{tf}(h, c_2) = \{e^0, f^0\}$.

Definition 2.4 (Last Fired Cluster Transition (LFCT) Abstraction). Let C be the cluster set of a net $\langle P, T, F\rangle$, $c \in C$, and H be the *history.* We define the *last fired cluster transition abstraction* $\alpha^{lfct} : H \times C \to T \cup \{\perp_i | i \in C\}$, where $\perp_i$ stands for no transition fired yet in cluster i:

$$\alpha^{lfct}(h, c) = \begin{cases} t, & \text{if } h = (\sigma; t; \gamma) \text{ and } t \in c \text{ and } \sigma \in T^* \text{ and } \gamma \in (T \backslash c)^* \\ \perp_c, & \text{if } h \in (T \backslash c)^* \end{cases}$$

The last fired cluster transition abstraction observes whether any transition in the cluster occurred in the history. If so, only the last occurred transition is counted. For instance, let $h = \langle a, x, y, b, a, a, b\rangle$, $c_1 = \{a, b, c\}$, and $c_2 = \{e, f\}$, then $\alpha^{lfct}(h, c_1) = b$ and $\alpha^{lfct}(h, c_2) = \perp_{c2}$. If $h = \varepsilon$, then $\alpha^{lfct}(h, c_1) = \perp_{c1}$ and $\alpha^{lfct}(h, c_2) = \perp_{c2}$.

2.3 Correlations

As we stated, decisions are often globally correlated. Future decision might be correlated with decisions made in the past. It implies that the estimation of $p(t, h)$ for a transition t in cluster $[t]$ depends on the abstracted history associated with $[t]$'s correlated decision. The formal definition of correlation is introduced as follows.

Definition 2.5 (Correlations). Let C be the *cluster set* of a net $\langle P, T, F\rangle$, $c_1, c_2 \in C$, and H be the *history.* The *correlation* is a function $R : C \times H \to C$ mapping c_1 to c_2, i.e. $R(c_1, h) = c_2$.

We introduce a short-hand notation for the correlated cluster $R([t])$ for cluster $[t]$. In the travel agency example, we assume the pickup booking decision is correlated with the hotel booking decision, denoted by $R([\mathit{PickUp}]) = [\mathit{ExpHotel}]$.

Assuming that decisions (clusters) are probably not based on the entire history, we propose to use abstractions introduced previously over the history to discover correlations from the logs. Types of correlations then differ in abstractions applied to discover correlations. For correlations given by the LFCT abstraction, only the last activity is considered. This is useful when only the most recent history is relevant for the decisions. For correlations given by the TF abstraction, the firing frequencies of some correlated cluster transitions are considered. For example, the more repair iterations have been executed on a product, the less likely it will be accepted.

Only when the correlation is discovered, $p(t, h)$ can be estimated. Since both the estimation of $p(t, h)$ and the correlation discovery are associated with abstractions, the abstraction for $p(t, h)$ estimation is given by the strongest correlation. The general approach and the implementation to discover correlations are discussed in Section 2.3.1 and Section 3.1 respectively.

2.3.1 Detecting Dependencies

In principal, correlations can either be provided by domain experts or be discovered from the logs, but usually such knowledge from domain experts is not available. In this section, we introduce a procedure to check whether the decision made in one cluster is significantly dependent on the decision made before based on log observations.

Abstraction value	$c_1.$	$y_1 \cdots$	y_j	$\cdots y_m$	$.c_n$
Range($\alpha(C_1)$)					
x_1 $\vdots$					
x_i			n_{ij}		
$\vdots$ x_l					
Range($\alpha(C_n)$)					

Table 2.1: Log observation matrix for clusters $c_1 \ldots c_n$. For simplicity, only the rows and columns for clusters c_x and c_y are shown. $x_1, \ldots, x_l$ stand for the abstractions of history associated with c_x and $y_1, \ldots, y_m$ stand for all the transitions in c_y. n_{ij} is the number of occurrences that abstraction value x_i appears in the prefix of transition y_j.

By analyzing the log, trace by trace, event by event, we first derive a matrix (see Table 2.1) with observations from the log associated with all clusters in the process. This kind of matrix is called *log observation matrix*. The rows of the *log observation matrix* are identified by the values of the range of all abstractions associated with each cluster and columns by the transitions from the set of clusters in the process. n_{ij} is the number of occurrences that abstraction value x_i appears in the prefix of transition y_j. Although the row identifiers of the log observation matrix differ per abstraction, the derivation of the log observation matrix and following procedures are applicable to each abstraction. Details of the log observation matrix derivation method are explained in Section 3.1.

For testing the dependency between two clusters, a contingency matrix is then derived from the *log observation matrix*. A *contingency matrix* is a matrix summarizing the conditional frequencies of two variables (clusters) and showing how these two variables (clusters)

Abstraction value	$y_1 \cdots$	y_j	$\cdots y_m$	
x_1				r_1
$\vdots$				$\vdots$
x_i		n_{ij}		r_i
$\vdots$				$\vdots$
x_l				r_l
	$k_1 \cdots$	k_j	$\cdots k_m$	N

Table 2.2: Contingency matrix for two clusters c_1 and c_2 for some abstraction α. $x_1, \ldots, x_l$ stand for the abstraction values of history associated with c_1 and $y_1, \ldots, y_m$ stand for all the transitions in c_2.

are dependent on each other. Suppose there are two random variables: X and Y, representing the abstractions of cluster c_1 and all transitions in the cluster c_2 respectively.

An illustration of a contingency matrix is shown in Table 2.2. The rows of the contingency matrix denoted by $x_1, \ldots, x_l$ stand for the abstractions of the history associated with c_1 and columns denoted by $y_1, \ldots, y_m$ stand for all the transitions in c_2. n_{ij} is the number of occurrences that abstraction x_i appears in the prefix of transition y_j, k_j is the total number of occurrences of transition y_j in the log, r_i is the total number of occurrences that abstraction x_i appears in the prefix of all transitions in c_2, and $\sum_{j=1}^{m} k_j = \sum_{i=1}^{l} r_i = N$.

Having this contingency matrix, we apply the chi-square test to assess the null-hypothesis that X and Y are independent. The test statistic,

$$\chi(X,Y) = \sum_{i=1}^{l} \sum_{j=1}^{m} \frac{(n_{ij} - E_{ij})^2}{E_{ij}}, \text{with } E_{ij} = \frac{r_i \cdot k_j}{N},$$

is used to determine whether the null-hypothesis is accepted or rejected. E_{ij} in the equation stands for the estimated number of occurrences that abstraction x_i appears in the prefix of transition y_j.

As $\chi(X,Y)$ is χ^2-distributed with the degree of freedom $(m-1)(l-1)$, if $\chi(X,Y)$ is larger than the chi-square random value with the degree of freedom $(m-1)(l-1)$ and the defined significance level, the null-hypothesis is rejected, i.e. c_1 is significantly dependent with c_2. For instance, let $\chi(X,Y) = 125.516$ with the degree of freedom $= 4$. Then we reject the null-hypothesis with 95% confidence, since the test statistic is bigger than 9.488, the critical value for the significance level 0.05 and the degree of freedom 4.

Note that during the dependency detection procedure, in order to discover all dependencies for any pair of clusters, from one log observation matrix, multiple contingency matrixes are derived so that multiple chi-square tests are carried out. Suppose that we have one log observation matrix and k clusters. To investigate k^2 correlations for k^2 cluster pairs, k^2 contingency matrixes and chi-square tests are required.

2.3.2 Selecting Correlations

After we have determined the dependent clusters, for one cluster c_1, it is possible to discover a set of clusters dependent on c_1. We could take all dependent clusters. However, not all dependent clusters are strongly correlated. For instance, for the travel agency process in Figure 1.1, suppose we have detected that cluster [*BookMore*] is dependent with [*Flight*], [*ExpHotel*], and itself. However, we also find out that the effect of booking flights or hotels on booking more packages is rather weak, i.e. it is a weak correlation. When significant changes occur on the booking flights or hotels, no significant changes are propagated to booking more packages of flights and hotels. By contrast, it is found out that the effect of booking more on itself is strong. Then, we consider [*Flight*] and [*ExpHotel*] is a weak correlation whereas we consider [*BookMore*] and itself as a srong correlation. For now, we propose only to consider the strongest correlated cluster.

To find out the strongest correlated clusters, we need to discover the strength of all dependencies. Chi-square test is not suitable as it only tests the significance of a dependency between two variables [8]. Instead, the correlation coefficient describes the strength of a dependency between two variables [8]. In principle, further analysis methods, such as data mining techniques or regression analysis can be conducted to determine the correlation coefficient. However, they are outside of the scope of this thesis. In this thesis, the strongest correlation in our simulation experiments is determined by the domain expert knowledge of the example process.

2.4 Determining $p(t,h)$

So far, having abstractions and the corresponding strongest correlation(s), we are ready to estimate transition probabilities $p(t,h)$ which are dependent on the strongest correlated cluster derived from the log. A log L is a finite set of possible histories H. This section defines $p(t,h)$ corresponding to the abstractions introduced in Section 2.2.

2.4.1 $p(t,h)$ for TF Abstraction

First, we estimate for every t the probability to choose t under the condition that s is the TF abstracted history of cluster $R([t])$. As long as the TF abstracted history associated with $R([t])$ is indeed observed in the log, i.e. there are historical observations in the log, we use the conditional probability to choose t. This conditional probability is called history-dependent conditional probability. For this purpose, we divide the frequency $\vartheta(t,s)$ of s being the abstracted history of cluster $R([t])$ occurring before t by the frequency $\xi(t,s)$ of s being the abstracted history of cluster $R([t])$ occurring before some transition in cluster $[t]$.

$$\vartheta(t,s) = \sum_{h \in L} L(h) \cdot |\{\tilde{h} | \exists \tilde{h}, \gamma \in T^* : h = (\tilde{h}; t; \gamma) \wedge \alpha^{tf}(\tilde{h}, R([t])) = s\}|,$$

$$\xi(t,s) = \sum_{h \in L} L(h) \cdot |\{\tilde{h} | \exists \tilde{h}, \gamma \in T^*, x \in [t] : h = (\tilde{h}; x; \gamma) \wedge \alpha^{tf}(\tilde{h}, R([t])) = s\}|.$$

In case the abstracted history associated with $R([t])$ is never observed before any transition from $[t]$ in the log, which results in $\xi(t,s) = 0$, we take the history-independent estimation of the probability of t by dividing the number of occurrences of t $\bar{\vartheta}(t)$ in the log by the number of occurrences of cluster $[t]$ $\bar{\xi}(t)$ in the log. This unconditional probability is called history-independent probability.

$$\bar{\vartheta}(t) = \sum_{h \in L} L(h) \cdot |\{\tilde{h} | \exists \tilde{h}, \gamma \in T^* : h = (\tilde{h}; t; \gamma)\}|,$$

$$\bar{\xi}(t) = \sum_{h \in L} L(h) \cdot |\{\tilde{h} | \exists \tilde{h}, \gamma \in T^*, x \in [t] : h = (\tilde{h}; x; \gamma)\}|.$$

In case there are no observations of transitions from cluster $[t]$ in the log, we assume that each transition in $[t]$ fires with equal probability. Then the full definition of $p^{tf}(t,h)$ is as follows:

$$p^{tf}(t,h) = \begin{cases} \dfrac{\vartheta(t, \alpha^{tf}(h, R([t])))}{\xi(t, \alpha^{tf}(h, R([t])))} & \text{if } \exists h \in L, x \in [t] : x \in h \wedge \xi(t, \alpha^{tf}(h, R([t]))) \neq 0; \\ \dfrac{\bar{\vartheta}(t)}{\bar{\xi}(t)} & \text{if } \exists h \in L, x \in [t] : x \in h \wedge \xi(t, \alpha^{tf}(h, R([t]))) = 0; \\ |[t]|^{-1} & \text{if } \forall h \in L, x \in [t] : x \notin h. \end{cases}$$

2.4.2 $p(t,h)$ for LFCT Abstraction

Following the idea to generalize $p^{tf}(t,h)$, we define $p(t,h)$ for the LFCT abstraction straightforwardly:

$$p^{lfct}(t,h) = \begin{cases} \dfrac{\vartheta(t, \alpha^{lfct}(h, R([t])))}{\xi(t, \alpha^{lfct}(h, R([t])))} & \text{if } \exists h \in L, x \in [t] : x \in h \wedge \xi(t, \alpha^{lfct}(h, R([t]))) \neq 0; \\ \dfrac{\bar{\vartheta}(t)}{\bar{\xi}(t)} & \text{if } \exists h \in L, x \in [t] : x \in h \wedge \xi(t, \alpha^{lfct}(h, R([t]))) = 0; \\ |[t]|^{-1} & \text{if } \forall h \in L, x \in [t] : x \notin h. \end{cases}$$

Note that the history-dependent conditional probability for the LFCT abstraction has a different interpretation from the TF abstraction. For the LFCT abstraction, the history-dependent conditional probability estimates the probability to choose t under the condition that s is the last fired transition in cluster $R([t])$ in the history, whereas in the definition of $p(t, h)$ for the TF abstraction s is a multiset of transitions of $R([t])$.

$$\vartheta(t,s) = \sum_{h \in L} L(h) \cdot |\{\tilde{h} | \exists \tilde{h}, \gamma \in T^* : h = (\tilde{h}; t; \gamma) \wedge \alpha^{lfct}(\tilde{h}, R([t])) = s\}|,$$

$$\xi(t,s) = \sum_{h \in L} L(h) \cdot |\{\tilde{h} | \exists \tilde{h}, \gamma \in T^*, x \in [t] : h = (\tilde{h}; x; \gamma) \wedge \alpha^{lfct}(\tilde{h}, R([t])) = s\}|,$$

where $\vartheta(t, s)$ denotes the occurrences that the last choice made in cluster $R([t])$ was observed before t in the log, and $\xi(t, s)$ denotes the last choice made in cluster $R([t])$ was seen before some transition in $[t]$.

Similarly, in case the last fired transition from $R([t])$ is never observed before any transition from $[t]$ in the log or no observations of transitions from cluster $[t]$ in the log, we use the history-independent probability and equal probability respectively to estimate $p(t, h)$.

The idea for determining $p^{tf}(t, h)$ and $p^{lfct}(t, h)$ can be straightforwardly transplanted for $p(t, h)$ definitions corresponding to any other abstractions. In the case that the abstracted history s associated with cluster $R([t])$ is observed before cluster $[t]$ in the log, the history-dependent probability $\frac{\vartheta(t,s)}{\xi(t,s)}$ is applied. In the case that the abstracted history s associated with cluster $R([t])$ is never observed before cluster $[t]$ in the log, the history-independent probability $\frac{\bar{\vartheta}(t,s)}{\bar{\xi}(t,s)}$ is applied instead. Otherwise (if $[t]$ is never observed in the log), the equal probability $|[t]|^{-1}$ is assumed.

Chapter 3

Implementation

In Chapter 2, we have introduced the methods to discover correlations and $p(t, h)$ for HD-SPNs. In this chapter, we introduce our approach to implement these ideas by process mining techniques. The essence of process mining is to extract knowledge from logs recorded by information systems and to analyze the underlying processes by using the knowledge from logs [18]. Process mining aims at improving the processes by providing techniques and tools for discovering process, control, data, organizational, and social structures from logs [18].

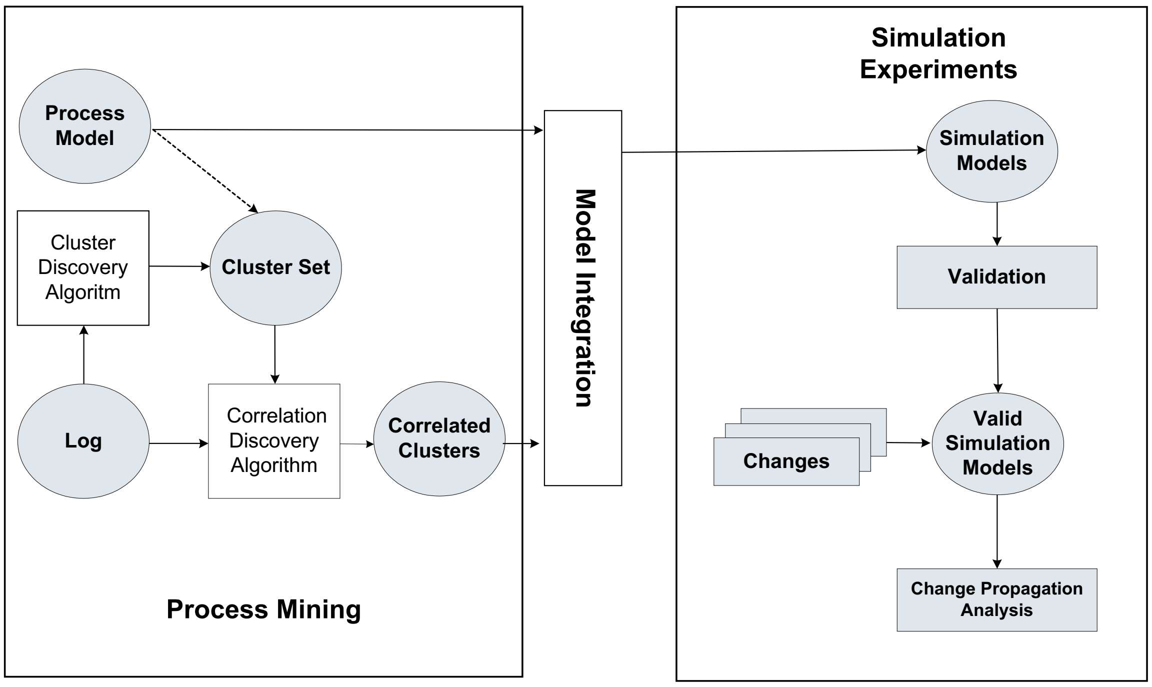


Figure 3.1: Overview about the process mining techniques used in this thesis.

In our research, process mining techniques assist us to discover correlations and estimate the history-dependent transition probabilities from the log. As depicted in Figure 3.1, with the help of the *cluster discovery algorithm* which is the existing work, the cluster sets are extracted from the log[1]. Afterwards, taking the cluster set and the log, the *correlation discovery algorithm* generates the log observation matrix and detects the correlated clusters. Based on the log observation matrix, transition probabilities $p(t, h)$ are calculated and integrated into HDSPNs.

For convenience, several concepts and notations are recalled and introduced first.

Let C be a cluster set, $c \in C$, and h be the history. The abstraction function $\alpha(h, c)$ keeps track of the current abstraction value of cluster c during the observation of the log. For every abstraction function, different abstraction values are kept. Implementation details about the LFCT abstraction and the TF abstraction are discussed in Section 3.1.2.

Let C be a cluster set, $c \in C$, e be a event, and h be the history. Let M be a $m \times n$ matrix with $M[i][j]$ the element at i^{th} row and j^{th} column. For every $M[i][j]$, the index j (index i) could be stored in a vector that you access by event names (a vector that you access by abstraction values), i.e. $M[Row(\alpha(h, c))][Column(e)]$.

3.1 Implementation of Correlations Discovery

The general algorithm to test whether the cluster is significantly dependent on a cluster is described in Section 3.1.1. As the general algorithm description does not specify implementation details associated with specific abstraction functions, these implementation details are further discussed in Section 3.1.2, in which the LFCT abstraction and the TF abstraction are given.

3.1.1 Algorithm Description

First of all, by analyzing the log, trace by trace, event by event, we try to derive the **Log Observation Matrix** denoted by O. Although some issues such as the abstraction values differ per abstraction, further procedures are applicable to any abstraction.

For every abstraction α, we first initialize the abstraction values for each cluster (step 1 in Table 3.1). For every event e in trace σ, we find column e in the log observation matrix (step 3.1) and check the rows to update, given by the current abstraction values for each cluster (step 3.2.1). For each row to be updated, we add an observation for event e (step 3.2.2). Meanwhile, for all clusters, we update abstraction values before reading the next

[1]Note that the cluster set can be obtained from the control flow as well.

Correlation Discovery Algorithm Part 1
inputs: Log, Cluster set, an abstraction type
outputs: Log observation matrix

```
function Log-Observer($Log, C, absType$) returns a log observation matrix $O$
inputs: $Log$, the log to be mined
            $C$, a set of clusters in the net
            $absType$, a selected abstraction function to discover correlations
variables: $O$, the log observation matrix, initially empty;
           $\vec{\alpha}$, a vector containing the abstraction values for all clusters;
               an abstraction value is accessed by cluster denoted by $\vec{\alpha}(c)$;
           $colIndex$, an integer indexing the column of $O$, initially 0;
           $rowIndex$, an integer indexing the row of $O$, initially 0;

1. $\vec{\alpha} \leftarrow$ initialize()*♯
2. For each trace $\lambda$ in $Log$ do
3. For each event $e \in \lambda$ do
   3.1. $colIndex \leftarrow$ getColumnIndex(e)*;
   3.2. For each cluster $c \in C$ do
      3.2.1. $rowIndex \leftarrow$ getRowIndex($\vec{\alpha}(c)$)*;
      3.2.2. $O[rowIndex][colIndex] \leftarrow O[rowIndex][colIndex] + 1$.
   3.3. End for each cluster $c \in C$ do
   3.4. $\vec{\alpha} \leftarrow$ updateAbs($absType$, e, $\vec{\alpha}$)*♯
4. End each event $e \in \lambda$ do
5. $\vec{\alpha} \leftarrow$ initialize()*♯
6. End each trace $\lambda$ in $Log$ do
7. Return $O$
```

Table 3.1: The pseudo code used to generate the log observation matrix. * User defined functions are called. $\sharp$ Differ for specific abstractions discussed in Section 3.1.2.

event in trace σ (step 3.4). Before reading the next trace, abstraction values for all clusters need be re-initialized, i.e. cleared and assigned with the initial values (step 5).

On the derived log observation matrix, we then apply **Chi-square Test**. In order to apply chi-square test to detect the dependencies between two clusters, the contingency matrix is required. Values (the number of occurrences) for the contingency matrix can be extracted from the log observation matrix. The sum of every row (r_i in Table 2.2) and the sum of each column (k_j in Table 2.2) can easily be added as well. Each extracted contingency matrix is the input for the apache library [2] that calculates the chi-square p-value for the corresponding dependency.

3.1.2 Implementation Details Associated with Specific Abstractions

As stated previously, some implementation issues differ per abstraction applied. The main differences lie in the initialization and updating of the abstraction values during the log observation matrix derivation phase. In this section, we address these specific implementations briefly with respect to the *transition frequency abstraction* and the *last fired cluster transition abstraction.*

3.1.2.1 Transition Frequency Abstraction

The TF abstraction function counts the frequencies of the transitions for the defined cluster. Recalling the definition of the transition frequency abstraction function, it projects the history h onto a cluster c and produces a *multiset* over cluster c (see Section 2.2). Consequently, the TF abstraction values should be in form of *multiset* over all clusters, i.e. for all $c \in C$, $\alpha^{tf}(h,c) = par(h \uparrow c)$, where C is a cluster set and h is the history.

For initialization (step 1 in Table 3.1), since the log is not read and no occurrence of any transition is observed (empty history, i.e. $h = \varepsilon$), for every cluster $c \in C$, $\alpha^{tf}(h,c) = par(\varepsilon \uparrow c)$.

When updating the TF abstraction for event e (step 3.4 in Table 3.1), we update the abstraction value given by e by adding 1 to the current frequency. The formal TF abstraction updating function for event $e \in c$ can be written as

$$\alpha^{tf}(h; e, c) = par(h; e \uparrow c) = par(h \uparrow c) + par(\langle e \rangle \uparrow c),$$

where ";" is the concatenation. This definition implies that when updating the TF abstraction given by some new event, we do not use the history for computing the new abstraction value every time. As $par(h \uparrow c)$ is known, only the new value $par(\langle e \rangle \uparrow c)$ needs computing.

For instance, $c_1 = \{a, b, c\}$, $c_2 = \{d, e\}$, then initially $\alpha^{tf}(\varepsilon, c_1) = \{a^0, b^0, c^0\}$ and $\alpha^{tf}(\varepsilon, c_2) = \{d^0, e^0\}$. When updating the TF abstraction for event b, as $b \in c_1$, we compute

$\alpha^{tf}(\langle b\rangle, c_1) = \{a^0, b^1, c^0\}$. The updated $\alpha^{tf}(\varepsilon; b, c_1) = \{a^0, b^0, c^0\} + \{a^0, b^1, c^0\} = \{a^0, b^1, c^0\}$ and $\alpha^{tf}(\varepsilon; b, c_2) = \{d^0, e^0\}$.

3.1.2.2 Last Fired Cluster Transition Abstraction

The LFCT abstraction function extracts the last fired cluster transition in the defined cluster. Recalling the definition of the LFCT abstraction function, the outcome is either a transition t or undefined $\perp_c$ (see Section 2.2). As a result, the LFCT abstraction values should be in form of a transition or undefined, i.e. for all $c \in C$, $\alpha^{lfct}(h, c) \in c \cup \{\perp_c\}$, where C is a cluster set and h is the history.

For initialization (step 1 in Table 3.1), since no cluster transition has been fired before, i.e. $h = \varepsilon$ and $\varepsilon \in (T \setminus c)^*$, for each cluster $c \in C$, $\alpha^{lfct}(h, c) = \perp_c$.

When updating the LFCT abstraction for event e (step 3.4 in Table 3.1), we update the abstraction value given by e by replacing the current cluster abstraction value by event e. The formal LFCT abstraction updating function for event $e \in c$ can be written as

$$\alpha^{lfct}(h; e, c) = e,$$

where ";" is the concatenation. This definition implies that when updating the LFCT abstraction given by some new event, we do not consider the history for computing the new abstraction value. Only the new event e is counted.

For instance, $c_1 = \{a, b, c\}$, $c_2 = \{d, e\}$, then initially $\alpha^{lfct}(\varepsilon, c_1) = \perp_{c_1}$ and $\alpha^{lfct}(\varepsilon, c_2) = \perp_{c_2}$. When updating the LFCT abstraction for event b, as $b \in c_1$, we replace the last fired cluster transition in cluster c_1 by event b, i.e. $\alpha^{lfct}(\varepsilon; b, c_1) = b$, while $\alpha^{lfct}(\varepsilon; b, c_2) = \perp_{c_2}$.

3.2 $p(t, h)$ Calculation and Implementation

In this section, we discuss the implementation of HDSPNs in CPN tools. Colored Petri Nets (CPNs) [10] is a discrete-event modeling language combining Petri nets with the functional programming language Standard ML [3]. This extended Petri net makes it possible to implement hierarchical structures and data dependencies. CPN tools [1], a high level Petri net tool, supports Colored Petri nets, and its simulator and monitors assist us to accomplish performance analysis.

First of all, we introduce the required additional elements in CPN models to support the history-dependent probability mechanism for HDSPNs. Afterwards, we propose a approach to calculate $p(t, h)$ from the log and to integrate the calculated $p(t, h)$ values into HDSPNs working with additional elements in Colored Petri Nets.

In HDSPNs, the global history is kept in a special place that contains a token with the history information, similarly to the implementation of History Nets [22]. Initially it contains a token containing the empty history. Every transition of the Petri net is then linked to the history place and transition firings update the history token by adding the information about the firing. Since our HDSPNs involve multiple abstractions, we keep the entire global history rather than an abstracted history for a case. To obtain the abstracted history, an ML implemented abstraction function is mapped onto the entire history and produces the corresponding abstracted history. Recall that the TF abstraction takes the history and a cluster and produces the multiset over the cluster (ML implementation see Appendix A.2) while the LFCT abstraction produces the last fired transition in the cluster (ML implementation see Appendix A.2).

To introduce a history-dependent probability mechanism, we need one random value for the entire cluster to make it work. Therefore, we add a fusion place random that contains a token whose value is randomly chosen from 0 to 999. Every firing of a transition results in updating the value of this token with a new random number. Within each cluster, transition guards capturing the probability mechanism are defined as follows: Let cluster c contain transition $t_1, \dots, t_n$ and given history h, the probability to choose transition t_i under condition that cluster c is chosen is $p(t_i, h)$. Then the guard for t_i is defined as

$$1000 \times \sum_{j=1}^{i-1} p(t_j, h) \leq \mathit{prob} < 1000 \times \sum_{j=1}^{i} p(t_j, h) \tag{3.2.1}$$

(for transition t_1, the lower border is 0), where *prob* is the value of the token on random. This definition of the guards ensures that in every cluster at every moment at most one transition can be enabled.

Now, let us take an example to see how $p(t, h)$ values, the global history place, and the fusion place random work together for HDSPNs. Figure 3.2 (a) depicts a net. In this net, there are clusters $[x] = \{x\}$, $[y] = \{y\}$, $[a_1] = \{a_1, a_2\}$ and $[b_1] = \{b_1, b_2, b_3\}$. The fusion place *History* connected to every transition is used to keep track of the firings. Every time a transition fires, the history is updated by adding the fired transition into the firing history (see the arc inscription "$b_1 :: \mathit{hist}$" in Figure 3.2 (b): b_1 is appended to the history *hist* when b_1 fires). The fusion place *Probability* connected to every transition in all clusters is used to generate a random value between 0 and 999. We assume that $p(b_1, h) = 0.1$, $p(b_2, h) = 0.35$, and $p(b_3, h) = 0.55$. By applying the transition guard definition 3.2.1, the guard for b_1 is defined as

$$0 \leq \mathit{prob} < 1000 \times p(b_1, h) = 100.$$

The guard for b_2 is defined as

$$1000 \times p(b_1, h) = 100 \leq \mathit{prob} < 1000 \times (p(b_1, h) + p(b_2, h)) = 450.$$

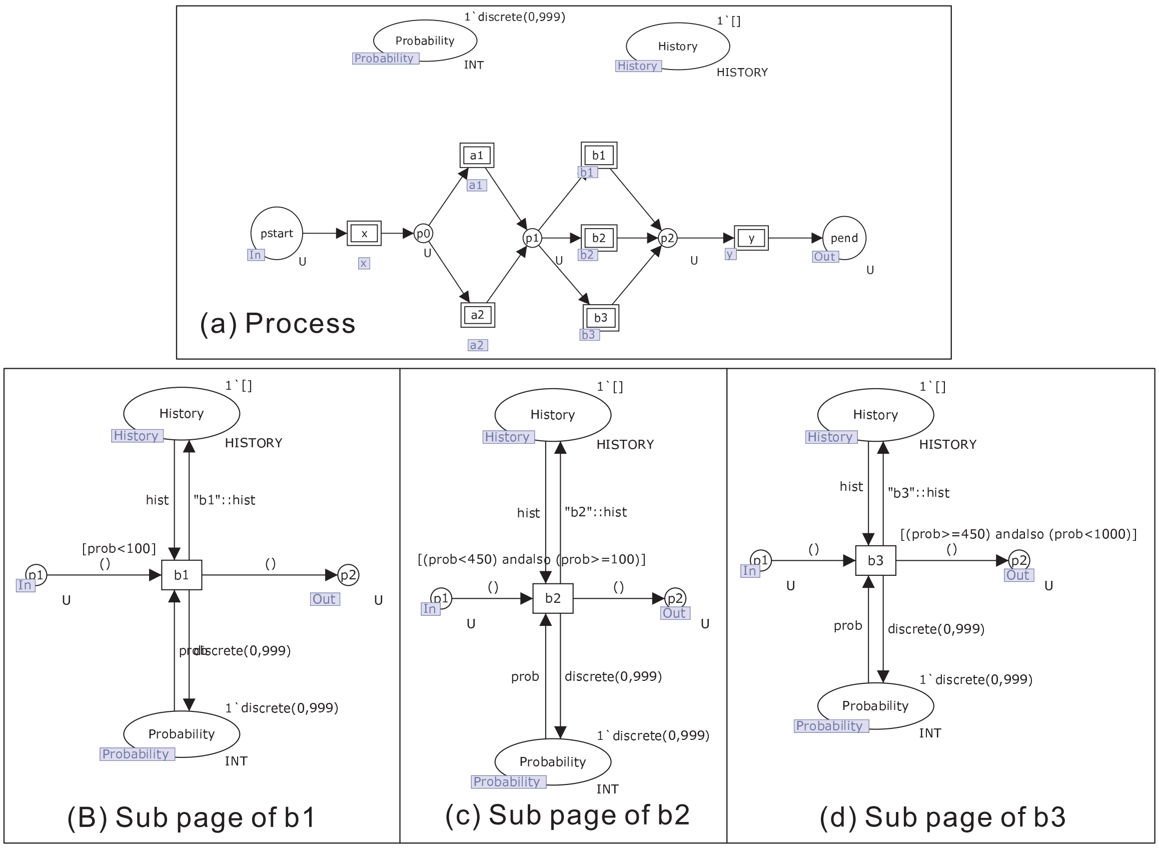


Figure 3.2: A sample of HDSPNs.

The guard for b_3 is

$$1000 \times (p(b_1, h) + p(b_2, h)) = 450 \le prob < 1000 \times (p(b_1, h) + p(b_2, h) + p(b_3, h)) = 1000.$$

So far we did not explain how $p(t, h)$ can be obtained from the log. The calculation of $p(t, h)$ starts with the log observation matrix derived from the log by the algorithm described in Section 3.1.1. To calculate $p(t, h)$ for transition t, the contingency matrix associated with $[t]$ and $R([t])$ is required. As values in the contingency matrix are exactly the occurrences $p(t, h)$ requires, $p(t, h)$ for HDSPNs can be easily calculated from the contingency matrix. Calculated $p(t, h)$ values are then implemented into an ML function in CPN tools of the HDSPN. The implemented ML function is placed in the guard on transition t in the HDSPN (see Figure 3.3). Every time t is enabled, the ML function takes the history and selects a probability for firing t according to the abstraction value on the history. In the example in Figure 3.2, ML functions can be defined that replace the constant values in the transition guards (i.e. 100, 450, and 1000) and that find the correct values given the history. Thus, the firing of the transition t is not only determined by the marking of the net but also by the history concerns.

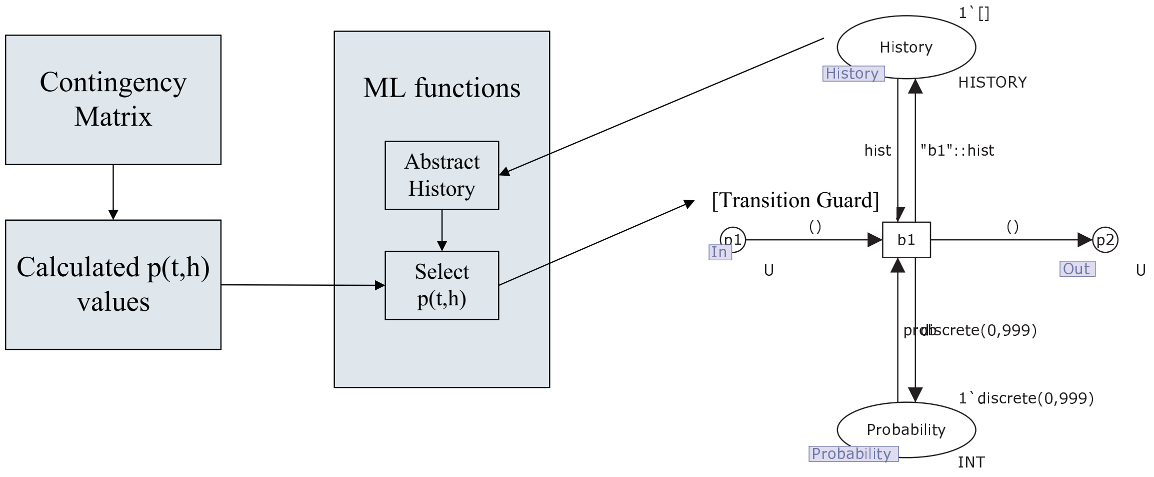


Figure 3.3: The dependencies among calculated $p(t,h)$, ML functions, and HDSPNs in Colored Petri Nets.

Let us recall the transition probabilities $p(t,h)$ defined in Section 2.4. Both for $p^{tf}(t,h)$ and $p^{lfct}(t,h)$ (for other abstractions as well), three possible probabilities under three situations would be considered, i.e. (1) a history-dependent conditional probability if the correlated abstraction value on the history is observed as the prefix of cluster $[t]$ in the log; (2) a history-independent probability if there are no correlated abstraction values as the prefix of $[t]$, but the log contains observations of cluster $[t]$; and (3) an equal probability if no observation about the entire cluster $[t]$ is found in the log.

Suppose that we have a transition t_j and its correlated cluster $R([t_j])$, the $p(t_j,h)$ calculation starts from the contingency matrix associated with $[t_j]$ and $R([t_j])$. Table 3.2 gives a table view of a contingency matrix associated with clusters $[t_j]$ and $R([t_j])$. The interpretation of the contingency matrix has already been introduced in Section 2.3.1. Here, based on this contingency matrix, we explain how three probabilities ((1), (2) and (3)) of $p(t_j,h)$ are obtained.

Suppose cluster $[t_j]$ with marked history h and abstraction value $\alpha(h,R([t_j]))=x_i$, then we determine $p(t_j,h)$ using the contingency matrix. The history-dependent conditional probability for t_j is formalized as

$$p(t_j,h)=\frac{n_{ij}}{r_i}, \text{if } \alpha(h,R([t_j]))=x_i,$$

$\alpha(h, R([t_j]))$	$t_1 \cdots$	t_j	$\cdots t_m$	
x_1				r_1
$\vdots$				$\vdots$
x_i		n_{ij}		r_i
$\vdots$				$\vdots$
x_l				r_l
	$k_1 \cdots$	k_j	$\cdots k_m$	N

Table 3.2: Contingency matrix for clusters $[t_j]$ and $R([t_j])$. $x_1, \ldots, x_l$ stand for the abstracted history associated with $R([t_j])$ and $t_1, \ldots, t_m$ stand for all the transitions in $[t_j]$. n_{ij} is the number of occurrences that abstraction x_i happened before transition t_j. r_i is the number of occurrences that abstraction x_i happened before cluster $[t_j]$. k_j counts the total number of occurrences of transition t_j regardless of the abstractions.

where h stands for the history of t_j and α is some abstraction function determined by the strongest correlation.

$\alpha^{tf}(h, [Flight])$	$Present$	$NoPresent$	$[Present]$	$p(Present, h)$
$\{Cruise^0, Flight^2\}$	0	380	380	0/380=0
$\{Cruise^0, Flight^3\}$	1	170	171	1/171=0.6%
$\{Cruise^0, Flight^4\}$	8	89	97	8/97=8.2%
$\{Cruise^0, Flight^5\}$	19	23	42	19/42=45.2%
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Total	3274	6726	10000	3274/6726=32.7%

Table 3.3: The extended contingency matrix with $p(t, h)$ calculations for clusters $[Present]$ and $[Flight]$. Note that the abstraction in this example is TF abstraction.

A concrete example for calculating $p(t, h)$ is shown in Table 3.3. Here, it is assumed that according to the TF abstraction cluster $[Present]$ is strongly correlated with cluster $[Flight]$ in the travel agency process in Figure 1.1. Column "$p(Present, h)$" lists the transition probabilities for transition *Present*. Given an abstracted history is $\{Cruise^0, Flight^3\}$, the corresponding $p(Present, h)$ is then equal to 0.6%, and so on and so forth.

If no abstraction value in the contingency matrix is found to be same as the abstracted history associated with $R([t_j])$ in HDSPNs, history-dependent conditional probability is not available and history-independent probability is used instead.

$$p(t_j, h) = \frac{k_j}{N}, \text{if } \forall i \in [1, l] : \alpha(h, R([t_j])) \neq x_i.$$

It is calculated exactly same as the transition probability calculation introduced in [15]. In Table 3.3, the history-independent probability for *Present* is "3274/6726=32.7%".

If the entire cluster $[t_j]$ is never observed in the log regardless of the abstraction values, i.e. $N = 0$, we define that all the transitions in cluster $[t_j]$ share the same probability to fire. Thus, in this situation, equal probability is preferred, which equals to the reciprocal of the cluster size $[t_j]^{-1}$. For instance, $[Present]$ cluster contains two elements and the equal probability is then $\frac{1}{2}$, i.e. 50%.

Since $p(t_j, h)$ values are dependent on the history, in the CPN implementation of HD-SPNs, ML functions are defined to select a $p(t_j, h)$ value for t_j according to the abstraction values on history. If the observed abstracted history is x_i, $i \in [1, l]$, then (1) is selected by ML functions. If the observed abstracted history doesn't exist in the log (contingency matrix), then (2) is selected. If the value of $N = 0$, then (3) is selected. The general structure of the function is defined in Table 3.4.

```
$p(t, h)$ **Function Implementation**
**Input: transition name, history**
**Output: probability value**

**function** p($t_j$, $h$) **returns** a probability
**inputs:** $t_j$, the transition to fire
            $h$, the history observed before $t_j$
**variable:** $p$, the transition probability, initially 0

1. If $[t_j]$ is not observed in the log ($N = 0$)
   1.1. $p \leftarrow [t_j]^{-1}$
2. End If 1
3. Else If $\alpha(h, R([t_j])) = x_i$
   3.1. $p \leftarrow \frac{n_{ij}}{r_i}$
4. End Else If 3
5. Else $p \leftarrow \frac{k_j}{N}$
6. Return $p$
```

Table 3.4: The pseudo code of $p(t, h)$ ML function. α is the abstraction function.

In Figure 3.4, we give a segment of the calculated and the ML implemented $p(t, h)$ for the TF abstraction for cluster $[Present]$ and cluster $[Flight]$ corresponding to the travel agency process. From the extended contingency matrix, we know that when the abstracted history is $\{flight^4, cruise^0\}$, the calculated $p(t, h)$ is 8.2% (see red circle 1 in Figure 3.4). The history-independent probability $p(t, h)$ is 32.7% (see red circle 2), which is the history-independent probability. In the ML code on the bottom, the "if" clause in function $TFprob()$ handles the condition which abstracted history is observed and "then" clause yields a corresponding $p(t, h) \times 1000$ for the transition guard in $[Present]$ (refer to

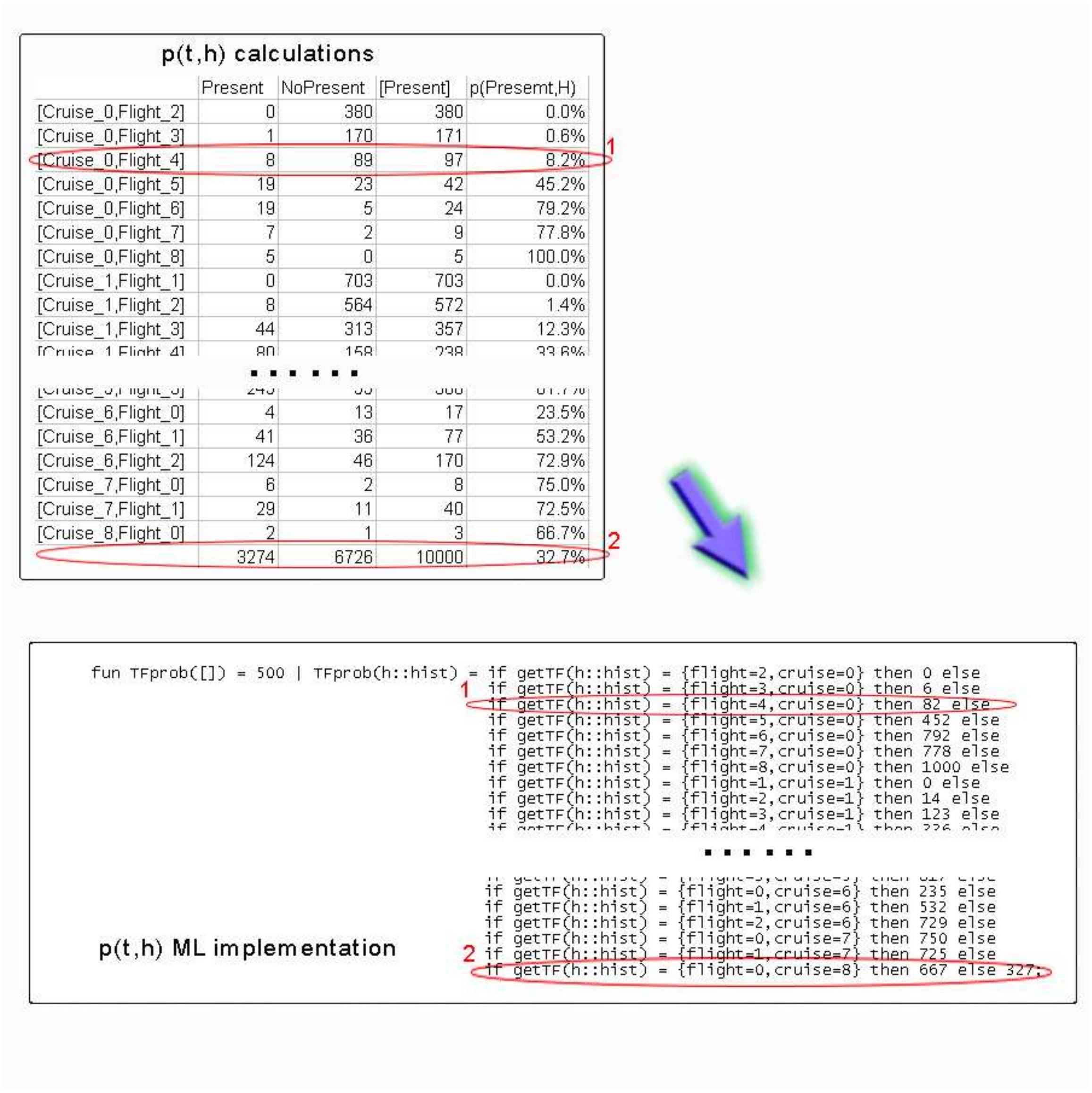
Equation 3.2.1). Note that in our experiments, only (1) and (2) are incorporated due to the size of the log[2].

3.3 ProM Implementation

We implemented our algorithm (see Section 3.1.1) to discover correlations from the log as a plug-in in the ProM framework[3]. Our plug-in is named "*cluster correlation miner*". ProM framework can read files in the XML format through its own component, which is able to deal with large data sets [21]. Our research requires mechanisms to read logs which are exactly ProM framework offers.

Another important feature of the ProM framework is that it allows for the interaction between a large number of plug-ins [21]. Due to this feature, our *cluster correlation miner* is chained with another miner called "cluster/decision miner". This miner outputs the cluster set which is one input of our "*cluster correlation miner*". Besides, *cluster correlation miner* is also chained with the log as another input. The last input, i.e. abstractions, is either selected from the GUI or from the default setting. Multiple selections of abstractions are allowed from GUI (see GUI selection in Figure 3.5). These chained objects can be defined in a macro (see Figure 3.5). The outputs of our plug-in are log observation matrices and the correlation results. The log observation matrix normally prints the whole log observation matrix for each abstraction. The correlation result shows the correlation detection results for every two clusters. The manual describing how to play with the "*cluster correlation miner*" is available in Appendix D.

[2]Enough observations ensure $N \neq 0$.
[3]www.processmining.org

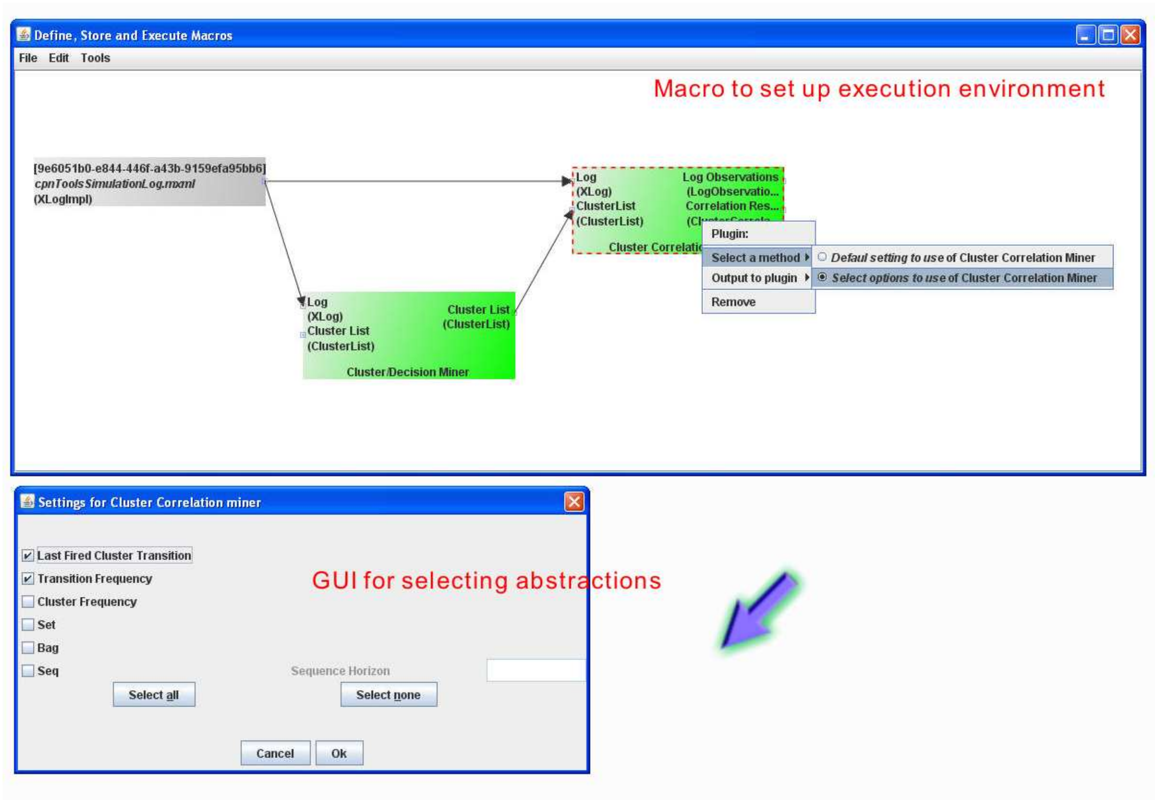


p(t,h) calculations

	Present	NoPresent	[Present]	p(Presemt,H)
[Cruise_0,Flight_2]	0	380	380	0.0%
[Cruise_0,Flight_3]	1	170	171	0.6%
[Cruise_0,Flight_4]	8	89	97	8.2%
[Cruise_0,Flight_5]	19	23	42	45.2%
[Cruise_0,Flight_6]	19	5	24	79.2%
[Cruise_0,Flight_7]	7	2	9	77.8%
[Cruise_0,Flight_8]	5	0	5	100.0%
[Cruise_1,Flight_1]	0	703	703	0.0%
[Cruise_1,Flight_2]	8	564	572	1.4%
[Cruise_1,Flight_3]	44	313	357	12.3%
[Cruise_1,Flight_4]	80	158	238	33.6%
......				
[Cruise_6,Flight_0]	4	13	17	23.5%
[Cruise_6,Flight_1]	41	36	77	53.2%
[Cruise_6,Flight_2]	124	46	170	72.9%
[Cruise_7,Flight_0]	6	2	8	75.0%
[Cruise_7,Flight_1]	29	11	40	72.5%
[Cruise_8,Flight_0]	2	1	3	66.7%
	3274	6726	10000	32.7%

```
fun TFprob([]) = 500 | TFprob(h::hist) = if getTF(h::hist) = {flight=2,cruise=0} then 0 else
                                          if getTF(h::hist) = {flight=3,cruise=0} then 6 else
                                          if getTF(h::hist) = {flight=4,cruise=0} then 82 else
                                          if getTF(h::hist) = {flight=5,cruise=0} then 452 else
                                          if getTF(h::hist) = {flight=6,cruise=0} then 792 else
                                          if getTF(h::hist) = {flight=7,cruise=0} then 778 else
                                          if getTF(h::hist) = {flight=8,cruise=0} then 1000 else
                                          if getTF(h::hist) = {flight=1,cruise=1} then 0 else
                                          if getTF(h::hist) = {flight=2,cruise=1} then 14 else
                                          if getTF(h::hist) = {flight=3,cruise=1} then 123 else
                                                           ......
                                          if getTF(h::hist) = {flight=0,cruise=6} then 235 else
                                          if getTF(h::hist) = {flight=1,cruise=6} then 532 else
                                          if getTF(h::hist) = {flight=2,cruise=6} then 729 else
                                          if getTF(h::hist) = {flight=0,cruise=7} then 750 else
                                          if getTF(h::hist) = {flight=1,cruise=7} then 725 else
                                          if getTF(h::hist) = {flight=0,cruise=8} then 667 else 327;
```

Figure 3.4: A segment of a extended contingency matrix with $p(t, h)$ calculations and ML implementation for the TF abstraction. Here we assume that cluster $[Present]$ is correlated with cluster $[Flight]$ in the travel agency process. *Circle 1* highlights an illustration of history-dependent conditional probability and *circle 2* is an illustration of history-independent probability. The highlighted *circle 1* in $p(t, h)$ calculation corresponds to the *circled 1* in the ML $TFprob$ function. So does highlighted *circle 2*.


Define, Store and Execute Macros
File Edit Tools
Macro to set up execution environment
[9e6051b0-e844-446f-a43b-9159efa95bb6]
cpnToolsSimulationLog.mxml
(XLogImpl)
Log
(XLog)
ClusterList
(ClusterList)
Log Observations
(LogObservatio...
Correlation Res...
Cluster Correlatic
Plugin:
Select a method
Output to plugin
Remove
Defaul setting to use of Cluster Correlation Miner
Select options to use of Cluster Correlation Miner
Log
(XLog)
Cluster List
(ClusterList)
Cluster List
(ClusterList)
Cluster/Decision Miner
Settings for Cluster Correlation miner
GUI for selecting abstractions
Select all
Select none
Cancel
Ok

Select a method: ( ) Defaul setting to use of Cluster Correlation Miner [✓ Select options to use of Cluster Correlation Miner]

- ✓ Last Fired Cluster Transition
- ✓ Transition Frequency
- ☐ Cluster Frequency
- ☐ Set
- ☐ Bag
- ☐ Seq

Sequence Horizon

Figure 3.5: The macro defined to execute the cluster correlation miner. Here, selecting abstractions from GUI is predefined.

Chapter 4

Simulation with HDSPNs

In this chapter, we introduce our procedures to conduct simulation experiments for evaluating process change propagation through simulation models. The goal of our simulation experiments is to verify that HDSPNs are able to propagate process changes through the process. Section 4.1 gives an overview of simulation experiments. Section 4.2 describes an extension of the running example in Figure 1.1. The running example is extended with data in order to obtain a more realistic example. This running example provides a log and a control flow for simulation modeling and further experiments. By using the log and the control flow, in Section 4.3, we will discuss the construction of different simulation models for the running example, i.e. (1) HDSPNs; and (2) Petri Nets with frequencies.

4.1 Experimental Overview

This section introduces a global setting of our simulation experiments.

Before we launch our simulation experiments, we need a process log and the corresponding control flow. In principal, there are a lot of logs and control flows available in the reality. However, by using these real processes and logs, only the current processes without process changes can be analyzed. For our research, we are interested in the propagation of the process change through the process, which is not in the logs yet. To observe the change propagation, process changes need to be introduced to the process. It is too risky and costly to ask a company or an organization to execute their processes in a changed setting to provide such logs. Therefore, we use an artificial example process on which we apply process changes and that provides the log. We extend the running example depicted in Figure 1.1 with data to play the role of this example process M_0. Details about this example process are in Section 4.2. On the log and the control flow, we construct simulation models.

As explained in Chapter 2, our hypothesis is that HDSPNs have the ability to propagate changes through business processes. In Section 4.3, we explain the implementation details of the HDSPN. In our experiments, we refer to the HDSPN model as M_1. We compare the HDSPN's capability of change propagation with existing work on extracting simulation models [15], which is proved to be unable to propagate changes. For this purpose, the Petri net with frequencies (denoted by M_2 in our experiments) is considered in experiments. All models involved in our experiments are summarized in Table 4.1.

Denotation	**Description**	**Data**	**History**	**Frequency**
M_0	Petri nets with data dependency	+	-	-
M_1	HDSPNs simulation model	-	+	+
M_2	Petri nets with frequency	-	-	+

Table 4.1: Summary of models for simulation experiments. + stands for the presence of the property and - for the absence.

Our simulation experiments start with validation without process changes in order to ensure the simulation models (M_1 and M_2) used in further experiments behave approximately as the example process (M_0) does before the change injection (See Figure 4.1). We first simulate M_0 *without* applying any process change repeatedly. During the repeat simulations, M_0 generates logs repeatedly. We randomly select a log among multiple logs. The log is used for mining correlations and $p(t, h)$ values for M_1. The frequencies for constructing M_2 are mined from the same log. Then, M_1 and M_2 are simulated repeatedly *without* applying any process change.

By running all models repeatedly *without* applying any process change, we record the transitions related to the performance indicators during the simulations. The performance of this running example is measured by decisions "Present" and "PickUp". Both decisions have complex dependencies with other parts of the process (see Section 4.2.1). Due to the financial concern, it is important to have an idea about the number of presents to order. Besides, estimating the number of "PickUp" is helpful for selecting the seasonal contracts with the taxi company for the pickup service. Therefore, we monitor the number of occurrences of these two decisions during the simulations by user-defined monitors in CPN tools.

When doing performance analysis, it is often necessary to collect statistically reliable data. As our example process is a finite process, i.e. a process which has an explicit start and an explicit end, replication simulations are preferred [6]. Replication simulations are simulation experiments with multiple observations on a group of individual cases. 50 replications are conducted to collect statistically reliable data as 30 replications are minimum [20]. In CPN tools, the function `CPN'Replications.nreplications` can be used to automatically run a given number of replications of simulations. To this end, a summary of experimental definitions for validation without changes is given in Table 4.2.

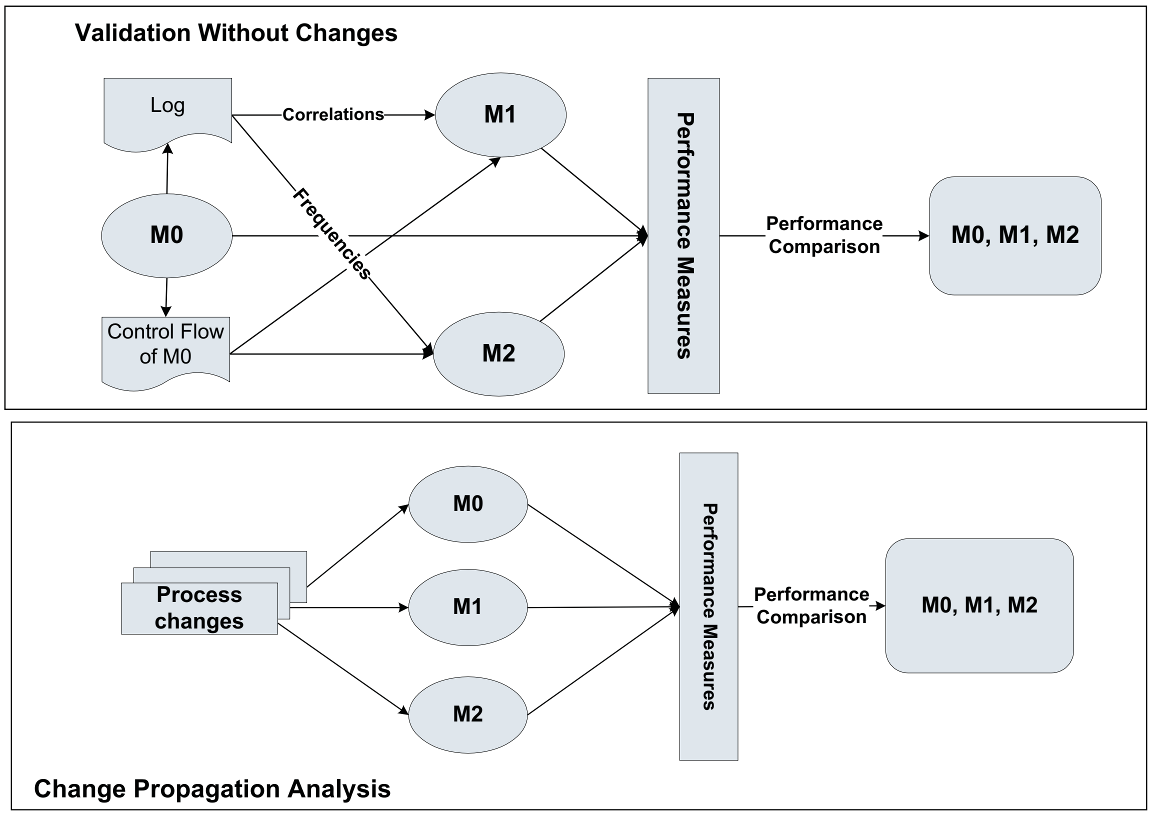


Figure 4.1: High-level Simulation experiment procedures.

By running simulation models repeatedly, we generate a sample of data on each performance indicator for each model. By applying statistic calculations, we can get the estimate and the confidence interval for each sample data. An estimate is a calculated approximation of the true performance that models would output [13]. Normally, the estimate is the mean value of the sample data. Confidence interval (CI) is an interval estimate for a data population [13]. As it is risky to estimate the true performance of the simulation model by a single value, an interval likely to include the true performance should be given. The formal calculation of CI is

$$[\overline{x} - \frac{z_{\alpha/2}\sigma}{\sqrt{n}}, \overline{x} + \frac{z_{\alpha/2}\sigma}{\sqrt{n}}],$$

where $\overline{x}$ is the sample mean, n is the size of observations, and $z_{\alpha/2}$ is the upper $100\alpha/2$ percentage point of the standard normal distribution at a significance level $\alpha = 95\%$.

We use CI analysis to compare the experimental results. By observing CIs of $\sharp Present$ and $\sharp PickUp$ for M_0, M_1, and M_2, the simulation model (M_1 or M_2) having the closer CI to M_0's CI is considered to be more precise.

Goal	investigate whether M_1 and M_2 approximately behave as M_0 without process changes
Object	occurrences of transitions “Present” and “PickUp”
Input Parameters	number of cases k number of replications n
Performance Indicators	frequencies of transition “Present” $\sharp Present$ frequencies of transition “PickUp” $\sharp PickUp$

Table 4.2: Summary of notions and notations used in the validation before introducing changes.

Further, change propagation analysis is conducted to verify whether M_1 is able to propagate process changes through the process well enough as M_0. The difference between change propagation analysis and validation is the introduction of process changes. Each individual process change is introduced into M_0, M_1, and M_2 respectively. In other words, in our change propagation analysis, only one process change is independently introduced into the process at the same time without incorporating with other process changes (if there are). Note that the remainder of the simulation models is unchanged, i.e. no new correlations, or frequencies are extracted from the log of the changed situation. Scenarios about what process changes are introduced into our process are given in Section 5.2.1. We record the number of occurrences of transition “Present” and transition “PickUp” after the injection of process changes. The experimental results are then analyzed by means of CI comparison as introduced above. The summary of experimental definitions is given in Table 4.3.

Goal	investigate whether M_1 approximately behave as M_0 after injecting process change;
Object	occurrences of transitions “Present” and “PickUp”;
Input Parameters	number of cases k number of replications n probability to fire transition “Flight”, *pFlight*;* (scenario 1) probability to fire transition “ExpHotel”, *pExpHotel*;* (scenario 2) probability to fire transition “MedHotel”, *pMedHotel*;* (scenario 2)
Performance Indicators	frequencies of transition “Present”, denoted as $\sharp Present$ frequencies of transition “PickUp”, denoted as $\sharp PickUp$.

Table 4.3: Summary of notions and notations used in the change propagation analysis after introducing changes. * These are parameters to which the changes will be applied.

4.2 Example Process and Its Model

As mentioned, we need to artificially construct an example process for our simulation experiments. To ensure the representativeness of actual processes, typical system properties, such as synchronization, sequentiality, parallelism, free choices, and data dependencies are required in this example process. In the travel agency process introduced in Chapter 1, synchronization, sequentiality, parallelism, and free choices are all involved. To make the stochastic behavior of the process more realistic and involved, we extend this running example with data. We next introduce data dependencies that influence the routing of a case, and its implementation in CPN tools.

4.2.1 Data Perspective of the Example Process

Let us recall our travel agency process (see Figure 4.2). Actually, the travel agency process depicted is an integration of processes of two parties, the external (clients) and the travel agency. Some activities are accomplished by clients and some are by the agency staff or the system. These processes spanning different parties could be modeled as an open net [9]. Since no effect would be given on the experiment results, we combine both processes into one process orchestration involving clients' behaviors and the system behavior. Table 4.4 shows the behaviors and the corresponding executors. The environment (client) behaviors and the system behaviors are further specified by the data perspective.

Transition	**Client**	**Company**
Register	$\surd$	
Flight/Cruise	$\surd$	
ExpHotel/MedHotel/LowHotel	$\surd$	
PickUp/NoPickUp	$\surd$	
BookMore/Confirm	$\surd$	
Installment/Noninstallment	$\surd$	
Present/NoPresent		$\surd$
Finish		$\surd$

Table 4.4: Transitions and their executors.

During the execution of activity *Register*, clients provide information about the number of additional bookings. After clients select transportation *Flight* or *Cruise*, the system adds the transportation costs to the total cost. During the selection of *ExpHotel*, *MedHotel*, or *LowHotel*, clients determine the number of nights to stay in the hotel, whose cost is added to the total cost by the system. Besides, the distances from the hotel to the airport/port and to the center are specified by the system. Normally, luxury hotels are expensive but

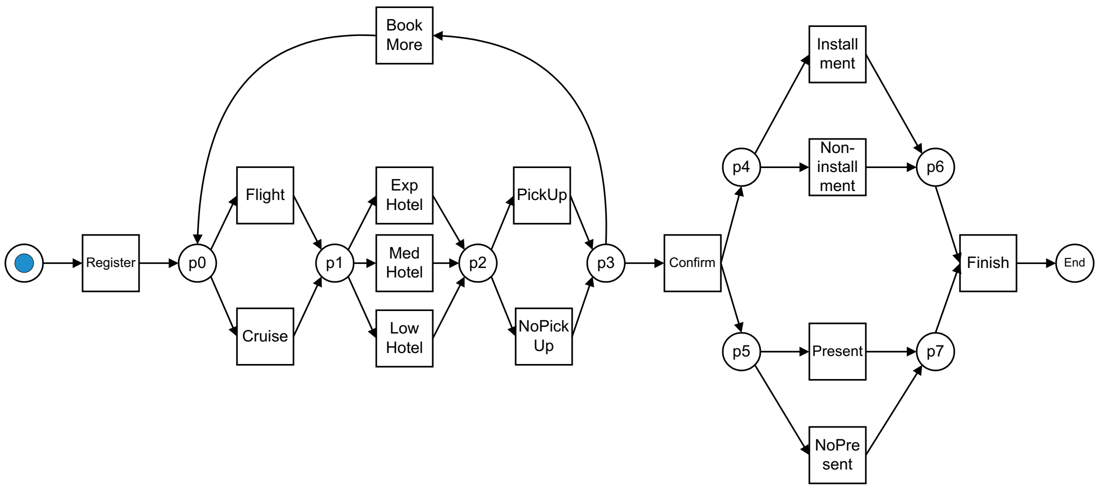


Figure 4.2: Example process: travel agency process.

near the center or the airport. Budget hotels are usually price-friendly but far away from the center and the airport. Middle class hotels balance between the cost and the location.

In addition to these transitions determining data, some decision rules are incorporated. During booking pickup services (*PickUp* or *NoPickUp*), clients determine to book or not mainly on the hotel's location and the hotel class. If the hotel booked is far from the center and the airport, clients who booked luxury hotels are more likely to prefer pickup services, while clients with budget hotels usually prefer walking. For the clients whose hotels are near both the center and the airport, pickup services are usually not necessary indeed. The *BookMore* activity is dependent on the number of the additional bookings predefined by clients initially. Finally, the present distribution (*Present*) is determined by the total cost. Only when the total cost exceeds 1000 euros, the agency distributes a present to the client. The dependencies between the decisions are summarized in Table 4.5.

4.2.2 CPN Implementation of the Example Process M_0

The example process is modeled in CPN tools in a hierarchical way as a Colored Petri net with data dependencies. The model is denoted by M_0.

Transitions	Decision dependencies	Client	Company
PickUp/NoPickUp	depending on the hotel's class and location	√	
BookMore/Confirm	depending on the number of the additional bookings predefined	√	
Present/NoPresent	depending on the total cost		√

Table 4.5: Decision dependencies and corresponding executors.

4.2.2.1 General Structure

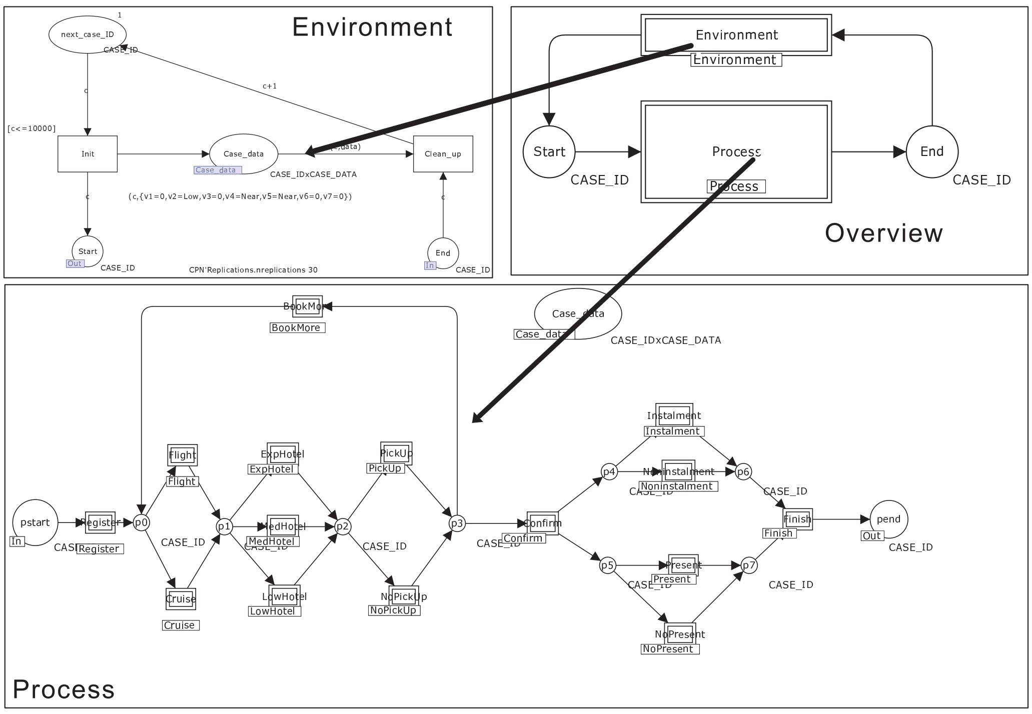


Figure 4.3: Hierarchical Structure of Example Process in CPN model

The top-level page in the hierarchical CPN model is shown in Figure 4.3. The environment generates cases and puts them into the *Start* place. Finally, it removes those that have reached the *End* place. Every case enters the "Process" page via the *Start* place and leaves the page via the *End* place. Note that there are two approaches to model the generation of cases, i.e. cases are simulated one by one or multiple cases are handled simultaneously. In our experiments, we assume that only one case is being handled at a time in the process. The next case is only handled when the previous one is finished. In this way, there is no need to check the case ID every time updating the data attributes. Simulation speed is enhanced without affecting the results.

For the data perspective, a separate token containing a record of case data attributes defined by the CASE_ID×CASE_DATA color set is created and initialized. The initial values represent default values for data attributes until they are explicitly specified. Note that since we simulate one case by one case, the CASE_ID color set is not necessarily defined. However, to make our models extendable, we keep CASE_ID color set and CASE_ID in CASE_ID×CASE_DATA color set. The place *CASE_DATA* is modeled as a fusion place as activities may need to read or write data attribute values on different pages in the hierarchical model.

Figure 4.3 shows the sub-page containing the example CPN model, which looks exactly like the original Petri net (see Figure 4.2). Every task on the process page has its own implementation. The remainder of this section illustrates the representation of certain process characteristics and inspecting or modifying data attributes in terms of CPN sub-pages.

4.2.2.2 Data and Data Dependency in M_0

Now we are going to incorporate the data and data dependencies into the model. Data attributes incorporated through M_0 are summarized in Table 4.6.

Attributes	**Type**	**Value Range**
ADDITIONAL LOOPS	Numerical	$[1, 7]$
COST	Numerical	$[0, 9999]$
CURRENT ITERATIONS	Numerical	$[2, 8]$
HOTEL	Enumeration	Lux/Med/Low
NIGHTS	Numerical	$[1, 5]$
DISTANCE_c	Enumeration	Far/Near
DISTANCE_p	Enumeration	Far/Near

Table 4.6: Data attributes involved in the model: where DISTANCE_c refers to the distance to center and DISTANCE_p refers to the distance to (air)port.

Every data attribute is assigned with values by some transition. Table 4.7 summarizes the transitions, corresponding data attributes and assigned values, as well as the corresponding ML Functions we defined to perform assigning values to data attributes. In registration *Register*, function $setA()$ selects a random numerical value from $[1, 7]$ for attribute *ADDITIONAL LOOPS*. It mimics the client behavior to register the number of additional bookings. In the action part of transition *Flight*, the function $setB1()$ selects a random numerical number for attribute *COST*. It simulates an action of adding transportation fee into the total cost. Similar configurations could be found in sub-process of *Cruise*.

Activity	Attributes	ML Functions	Values
Register	ADDITIONAL LOOPS	$setA()$	discrete(1,7)
Flight	COST	$setB1()$	discrete(120,300)
Cruise	COST	$setB2()$	discrete(50,80)
ExpHotel	COST DISTANCE_c DISTANCE_p HOTEL NIGHTS	$setC1()$	discrete(150,300) 50% Near 50% Near Lux discrete(1,5)
MedHotel	COST DISTANCE_c DISTANCE_p HOTEL NIGHTS	$setC2()$	discrete(80,150) 40% Near 40% Near Med discrete(1,5)
LowHotel	COST DISTANCE_c DISTANCE_p HOTEL NIGHTS	$setC3()$	discrete(30,80) 10% Near 10% Near Low discrete(1,5)
PickUp	COST	$setE()$	10
BookMore	CURRENT ITERATIONS	$setMore()$	+1*
Confirm	COST CURRENT ITERATIONS	$setD1()$ $setMore()$	discrete(2,10) +1*

Table 4.7: Parameters for data provision rules. * stands for adding 1 to the old value of CURRENT ITERATIONS.

More complex implementations of provision of case data can be seen in the action part of transition *ExpHotel* (in Figure 4.4). Function $setC1()$ writes five values into five data attributes, i.e. COST, HOTEL, DISTANCE_c, DISTANCE_p, and NIGHTS. Two fusion places *Probability1* and *Probability2* in the sub page *ExpHotel* (see Figure 4.4 (a)) are used to control the probabilities to determine the DISTANCE type ("Near" or "Far") for attributes DISTANCE_c and DISTANCE_p in function $setC1()$. The idea behind is to determine the distance with certain probabilities. All the modified data values are stored in the CASE_ID×CASE_DATA token. Similar implementations are in sub-pages of *MedHotel* and *LowHotel*. Functions $setD1()$ and $setE()$ select a random numerical value and a constant value respectively for data attribute *COST* when *PickUp* and *Confirm* are fired. It simulates an action of adding the pickup service fee and the confirmation service fee into the total cost.

The decision rules in Table 4.8 depend on data attributes (see in Table 4.6) evoked by tasks *Register*, *Flight*, *Cruise*, *ExpHotel*, *MedHotel*, and *LowHotel*. These data dependencies

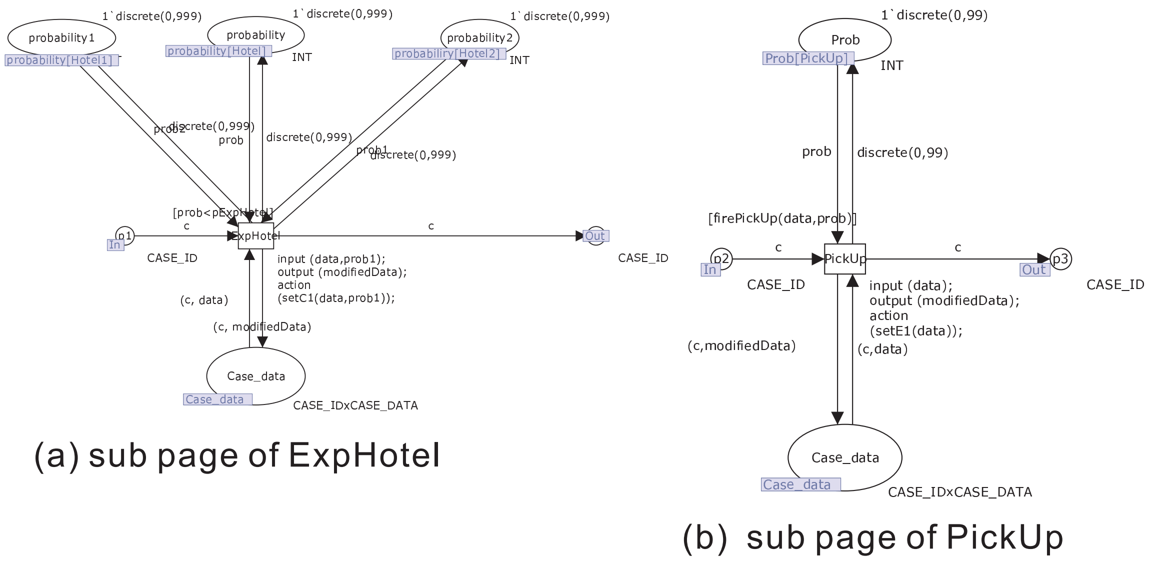


Figure 4.4: Data dependencies Implementation.

are modeled in sub-pages by transition guards. If the transition is enabled from a marking perspective, it additionally needs to satisfy the given guard condition to be fired. When distributing presents, the guard [♯v3 data>1000] restricts that presents can only be sent when the cost is above 1000 euros. "♯v3 data" in the guard refers to the data attribute COST, i.e. the total cost.

There are some more complicating implementations of data dependencies, such as functions $PickUpProb()$ and $firePickUp()$ in the transition guard given in Figure 4.4 (b). Function $PickUpProb()$ determines the probability for booking a pickup service corresponding to the distance to the center and to the airport, as well as the hotel class. The intuition is that not all clients would like to have the pickup service. Clients booked different hotels (luxury, middle-class, or budget) with different distances to the center and the airport have different demands for pickup services. A boolean function $firePickUp()$ compares whether the random number generated by the fusion places *Prob* is smaller than the number $PickUpProb()$ function determined, in other words, whether *PickUp* can fire. *PickUp* fires only when it is enabled from a marking perspective and $firePickUp()$ returns true. Similar implementation can be seen in $setMore()$ and $fireMore()$ functions.

Detailed implementations of ML functions mentioned above, as well as the other functions involved in M_0, can be found in the CPN source code in Appendix A.1.

Activity	**Variables**	**Rule**	**Function**
BookMore	ADDITIONAL LOOPS, CURRENT ITERATIONS	CURRENT ITERATIONS < ADDITIONAL LOOPS	$setMore()$, $fireMore()$
Confirm	ADDITIONAL LOOPS, CURRENT ITERATIONS	CURRENT ITERATIONS ≥ ADDITIONAL LOOPS	$setMore()$, $fireConfirm()$
PickUp	DISTANCE_c, DISTANCE_p, HOTEL	(Far, Far) with 85%/65%/50% (Lux/Med/Low), (Far, Near) with 60%/35%/20% (Lux/Med/Low), (Near, Far) with 60%/35%/20% (Lux/Med/Low), (Near, Near) with 5%/5%/5% (Lux/Med/Low)	*PickUpProb*(), *firePickUp*()
NoPickUP	DISTANCE_c, DISTANCE_p, HOTEL	$1 - PickUpProb()$	*PickUpProb*(), $fireNoPickUp()$
Present	COST	COST $\geqslant$ 1000	/
NoPresent	COST	COST < 1000	/

Table 4.8: Parameters for data dependency rules.

4.3 CPN Implementation of HDSPN with Correlations M_1

This section illustrates a HDSPN with correlations as a Colored Petri net for the example process. Two correlations are incorporated, i.e. an LFCT abstraction strongest correlation and a TF abstraction strongest correlation. According to the domain expert knowledge of the example process, we deduce that [*PickUp*] is strongly correlated with [*ExpHotel*] under the LFCT abstraction and [*Present*] strongly correlated with [*ExpHotel*] under the TF abstraction (see Table 4.9).

Cluster	**Correlated Cluster**	**Abstraction Type**
[*PickUp*]	[$ExpHotel$]	LFCT abstraction
[*Present*]	[$ExpHotel$]	TF abstraction

Table 4.9: Clusters and their correlated clusters.

As described in Section 3.2, two additional elements are required to support the history-dependent probability mechanism for HDSPNs (see in Figure 4.5). A fusion place "History" containing a token with the history information is used to keep track of the global history. Every transition connected to "History" updates the history token by adding the firing information when it fires. Another fusion place random "Probability" is added, which contains a token whose value is randomly chosen from 0 to 999. Motivations to introduce these two fusion places are discussed in Section 3.2.

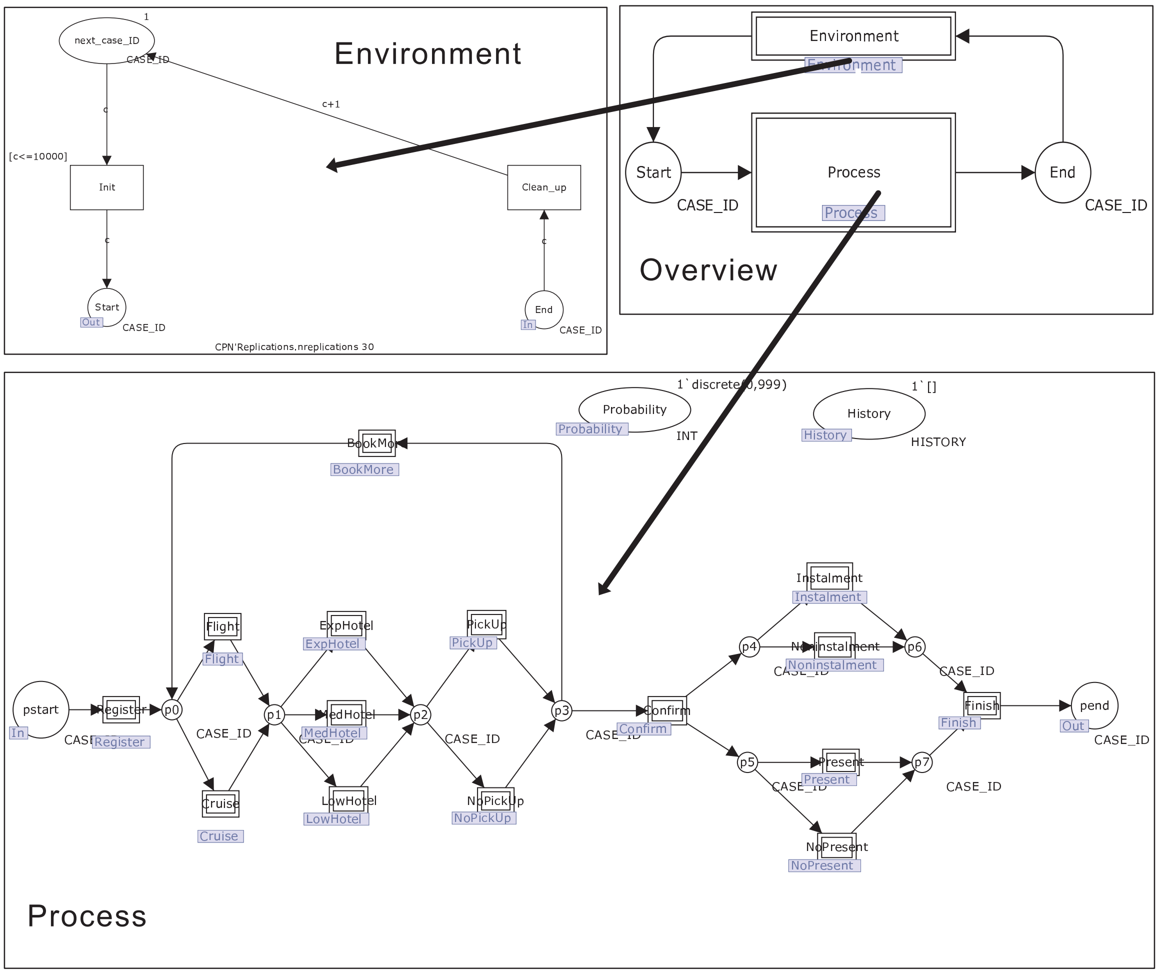


Figure 4.5: Hierarchical Structure of M_1 HDSPNs in CPN model.

With the help of the *cluster correlation miner* implemented in ProM framework, the $p^{tf}(Present, h)$ values and the $p^{lfct}(PickUp, h)$ values are calculated from the log generated by M_0 by following the procedures to calculate $p(t, h)$ values proposed in Section 3.2. The calculated $p^{tf}(Present, h)$ values are partially summarized in Table 4.10 and complete $p^{lfct}(PickUp, h)$ values are in Table 4.11.

Calculated $p^{tf}(Present, h)$ values are integrated into the ML function *TFprob2(history)*. According to the observed history, *getTF(history)* function abstracts the desired TF abstraction value on the history associated with the correlated cluster $[ExpHotel]$. Then, *TFprob2(history)* selects a calculated $p^{tf}(Present, h)$ value according to the abstraction value generated by *getTF(history)* function. The boolean function *firePresent(pro, history)* compares whether the random number generated by the fusion places *Probability* is smaller

$\alpha^{tf}(h, [ExpHotel])$	$p(Present, h)$
$\vdots$	$\vdots$
$\{ExpHotel^0, MedHotel^1, LowHotel^3\}$	0
$\{ExpHotel^0, MedHotel^1, LowHotel^4\}$	3.1%
$\{ExpHotel^0, MedHotel^1, LowHotel^5\}$	22.2%
$\{ExpHotel^0, MedHotel^1, LowHotel^6\}$	0
$\vdots$	$\vdots$
$History - independent$	32.7%

Table 4.10: Partial calculated $p(t, h)$ values for clusters $[Present]$ and $[ExpHotel]$. Note that the abstraction in this example is the TF abstraction. Due to the size of the results, only part of them is shown.

$\alpha^{lfct}(h, [ExpHotel])$	$p(PickUp, h)$
$ExpHotel$	52.9%
$MedHotel$	41.4%
$LowHotel$	44.1%
$History - independent$	46.1%

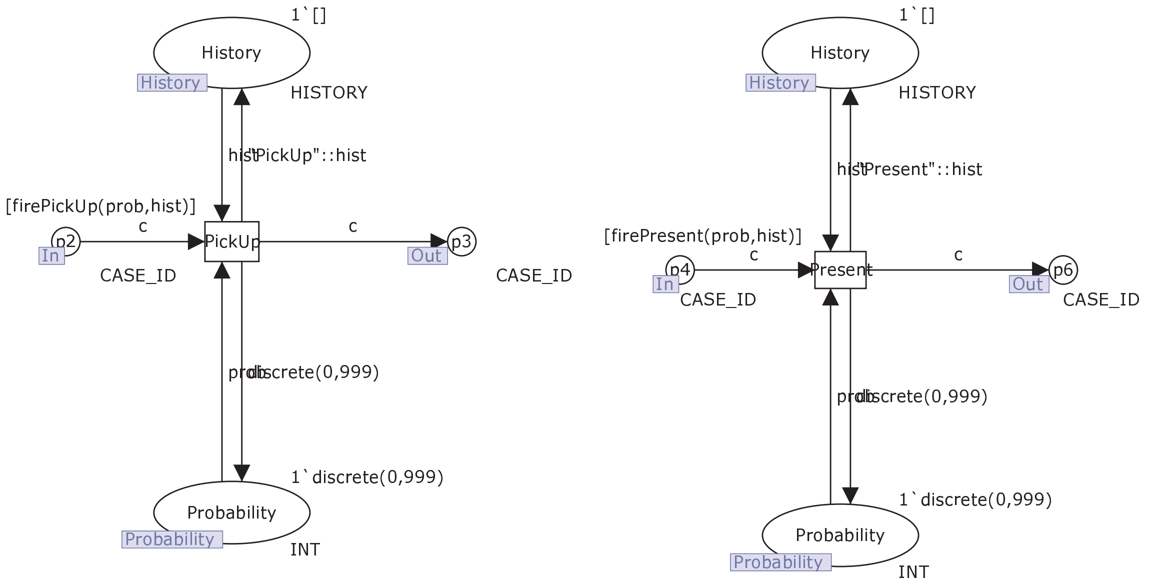
Table 4.11: $p(t, h)$ values for clusters $[PickUp]$ and $[ExpHotel]$. Note that the abstraction in this example is the LFCT abstraction.

than the number *TFprob2(history)* function selected. The boolean function is directly placed in the transition guard of *Present* to control the firings (see in Figure 4.6 (b) and (c)). *Present* fires only when it is enabled from a marking perspective and *firePresent(pro, history)* returns true. Similar implementations are applied to the LFCT abstraction correlation between $[ExpHotel]$ and $[PickUp]$. The ML functions supporting HDSPNs mechanism in M_1 are summarized in Table 4.12 and Table 4.13.

Function	**Action**
getTF(history)	Returns the TF abstracted history of cluster [ExpHotel]
TFprob2(history)	Returns $p^{tf}(Present, history)$
firePresent(prob,history)	Returns true if 0 $\leq$ prob $<$ $p^{tf}(Present, history)$

Table 4.12: The TF abstraction correlation for cluster [Present]; "history" is a list; "prob" is a random value.

Details about the ML functions in Table 4.13 and Table 4.12 as well as other ML functions involved in M_1 can be found in the CPN source code in Appendix A.2.



(a) Sub page of PickUp

(b) Sub page of Present

Figure 4.6: Correlation implementations for M_1. It is assumed that [Present] is strongly correlated with [ExpHotel] under the TF abstraction and [PickUp] is strongly correlated with [ExpHotel] under the LFCT abstraction.

4.4 CPN Implementation of Petri Nets with Frequencies M_2

To construct a Petri Net with frequencies to simulate our travel agency process as a comparison with the HDSPN, we adopt the approach introduced in [15]. Complex dependencies are replaced by probability distributions. A firing probability is assigned to each alternative of the decision to mimic the decision making by users. These history-independent probability for all branches of the decisions can be derived from the same log, which is used for constructing M_1, by means of the approach introduced in 3.2. The history-independent probabilities incorporated in M_2 are summarized in Table 4.14.

In M_2, the hierarchical CPN model looks similar to the M_1's hierarchy. One difference is

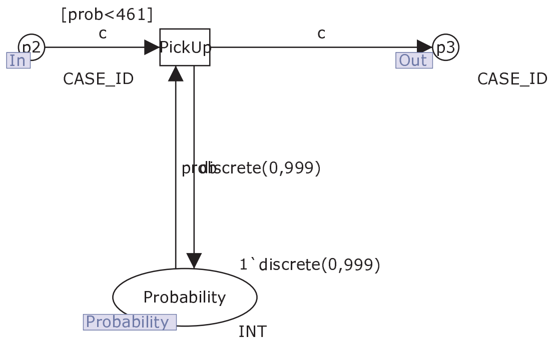
Function	**Action**
getLFCT(history)	Returns the last fired transition of [ExpHotel] in the history
LFCTprob(history)	Returns $p^{lfct}(PickUp, history)$
firePickUp(prob,history)	Returns true if $0 \leq$ prob $< p^{lfct}(PickUp, history)$

Table 4.13: The LFCT abstraction correlation for cluster [PickUp]; "history" is a list; "prob" is the random value.

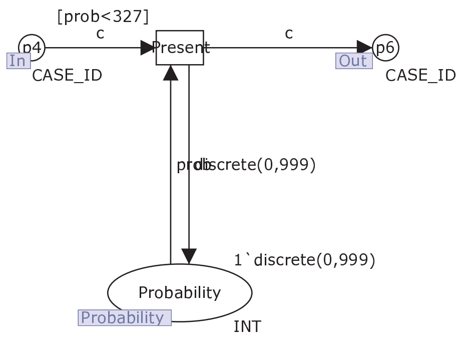
Cluster	**Transition**	**Percentage**
$[Flight]$	$Flight$	50%
	$Cruise$	50%
	$ExpHotel$	33.3%
$[ExpHotel]$	$MedHotel$	33.3%
	$LowHotel$	33.3%
$[PickUp]$	$PickUp$	46.1%
	$NoPickUp$	53.9%
$[BookMore]$	$BookMore$	90%
	$Confirm$	10%
$[Present]$	$Present$	32.7%
	$NoPresent$	67.3%
$[Installment]$	$Installment$	50%
	$Noninstallment$	50%

Table 4.14: History-independent probabilities in M_2.

the absence of the fusion place "History". M_2 does not consider the correlations; therefore there is no need to keep track of transitions' firings. Instead, static probabilities are assigned to the transition guards. Differing from M_1 that uses many ML functions for determining $p(t, h)$ in the guards, guards in M_2 only involve constant values (see Figure 4.7). *Present* or *PickUp* fires only when it is enabled from a marking perspective and the guard returns true.



(a) Sub page of PickUp



(b) Sub page of Present

Figure 4.7: CPN sub-page implementations for M_2. The probability to fire *PickUp* is 46.1%. The probability to fire *Present* is 32.7%

Chapter 5

Simulation Results with HDSPNs

This chapter presents the results for the experiments described in Chapter 4. The results for the experiments without applying change are presented in Section 5.1. Section 5.2 presents the results for the experiments after applying change.

5.1 Validation Results without Process Changes

Before we introduce process changes to simulation experiments, we validate the behavior of the models in absence of change. We define several sets of strongest correlations to validate and verify the change propagation of HDSPNs (see Table 5.1).

Correlation Set	Correlated Cluster	Abstraction Type
1	$R([PickUp]) = [ExpHotel]$	LFCT abstraction
	$R([Present]) = [Flight]$	TF abstraction
	$R([BookMore]) = [BookMore]$	TF abstraction
2	$R([PickUp]) = [ExpHotel]$	LFCT abstraction
	$R([Present]) = [ExpHotel]$	TF abstraction
	$R([BookMore]) = [BookMore]$	TF abstraction
3	$R([PickUp]) = [ExpHotel]$	LFCT abstraction
	$R([Present]) = [BookMore]$	TF abstraction
	$R([BookMore]) = [BookMore]$	TF abstraction

Table 5.1: Clusters and their strongly correlated clusters.

According to the data perspective of the example process, we know that the more flights, more expensive hotels, or more travel packages are booked, the more presents are distributed. Thus, we deduce that cluster $[Present]$ can be strongly correlated with cluster $[Flight]$, $[ExpHotel]$, or $[BookMore]$ under the TF abstraction. As determining the strength of the correlation is out of the scope of this thesis, we try these different correlations one by one in the experiments. As the more travel packages have been booked, the more unlikely it is that another travel package in the next round would be booked. Therefore, we deduce that cluster $[BookMore]$ is strongly correlated with itself under the TF abstraction. Besides, we also know that the decision to book a pickup service is only affected by the hotel booked. Thus, we determine that cluster $[PickUp]$ is strongly correlated with cluster $[ExpHotel]$ under the LFCT abstraction.

Here we present the validation results for M_1 with a TF abstraction correlation between the clusters $[Present]$ and $[Flight]$, a TF abstraction correlation between the clusters $[BookMore]$ and $[BookMore]$, and an LFCT abstraction correlation between the clusters $[PickUp]$ and $[ExpHotel]$. Validation results for M_1 with other correlation sets can be found in Appendix B. To determine whether a deviation between two values is significant or negligible, we define a rule as follows.

Rule 5.1. *Let v be the actual value and v_{appro} be the approximate value. when the relative deviation definition $\rho = \frac{|v-v_{appro}|}{|v|}$ is smaller then 2%* [1]*, we say the deviation is negligible otherwise the deviation is significant.*

In our travel agency process, we initially assume that the travel agency process has 1000 individual cases. By running 1000 cases with 50 replications on M_0, M_1 and M_2 respectively, the validation results are represented in Figure 5.1. Figure 5.1 shows that both M_1 and M_2 closely approximate the fraction of presents and pickups of the example process M_0, with a relative deviation (ρ) of 0.9% and 0.6% respectively for the presents and a relative deviation (ρ) of 0.2% and 0.09% for pickups. They are all negligible according to Rule 5.1.

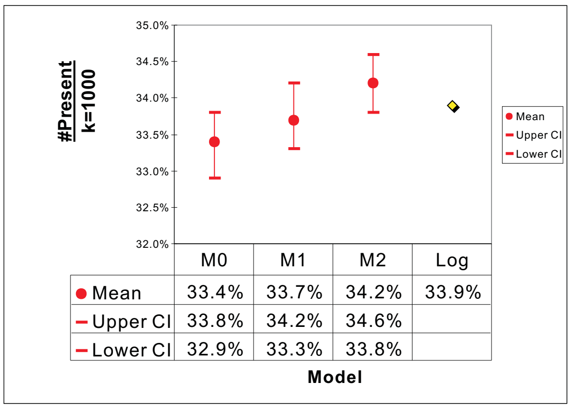
In addition, as an extra check, we further investigate the mined $p(t,h)$ values for cluster $[PickUp]$, which is correlated with cluster $[ExpHotel]$ under the LFCT abstraction. These mined $p(t,h)$ values are compared with the estimated $p(t,h)$ values from statistical calculations[2]. Here, we give an example how the estimated $p(t,h)$ values are calculated. Referring to the data dependency rules in Table 4.8, the estimated $p(PickUp, ExpHotel) = p((Far, Far)|Lux) + p((Near, Far)|Lux) + p((Far, Near)|Lux) + p((Near, Near)|Lux) = 50\% \times 50\% \times 85\% + 50\% \times 50\% \times 60\% + 50\% \times 50\% \times 60\% + 50\% \times 50\% \times 85\% = 52.5\%$. Other estimated $p(t,h)$ values and mined $p(t,h)$ values are in Table 5.2.

Surprisingly, deviations exist between estimated and mined $p(t,h)$ values. Especially when the last fired cluster transition is $MedHotel$, the relative deviation is around 4.6%

[1]The threshold for the relative deviation is user-defined. In this thesis, 2% is selected.

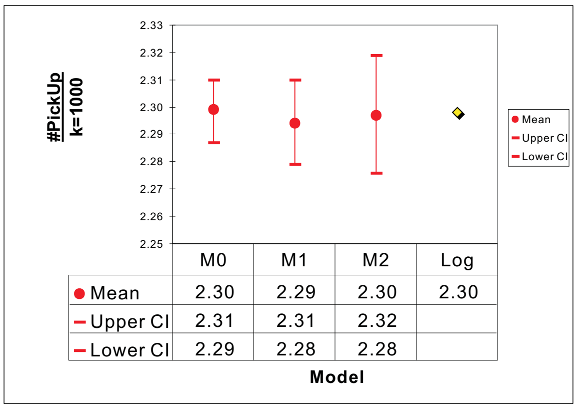
[2]According to the data dependency between $[PickUp]$ and $[ExpHotel]$, it is possible to estimate $p(t,h)$ by means of conditional probability calculation.

Number of cases:1000
Validation Result: No process change has introduced
Correlations: [Present]&[Flight]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]



	M0	M1	M2	Log
● Mean	33.4%	33.7%	34.2%	33.9%
− Upper CI	33.8%	34.2%	34.6%	
− Lower CI	32.9%	33.3%	33.8%	

(a) TF abstraction: R([Present])=[Flight]



	M0	M1	M2	Log
● Mean	2.30	2.29	2.30	2.30
− Upper CI	2.31	2.31	2.32	
− Lower CI	2.29	2.28	2.28	

(b) LFCT abstraction: R([PickUp])=[ExpHotel]

Figure 5.1: Validation result with 1000 cases and 50 replications without applying process changes. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. Note that we use a fraction scaling, which is $\frac{indicator}{k=1000}(\times 100\%)$.

(see in Table 5.2), which is significant according to Rule 5.1. Significant deviations would impact on the precision of the change propagation analysis.

These deviations are probably caused by the number of cases, which fails to provides sufficient observations for mining $p(t,h)$. Table 5.3 shows the newly mined $p(t,h)$ values after increasing the number of cases to 10000. It is explicit that the mined $p(t,h)$ values from a 10000-case log are closer to the estimated $p(t,h)$ values. The relative deviations which are smaller than 2% can are negligible in our experiments. Therefore, the following experiments are simulated with 10000 cases for all models.

To save simulation time, the number of replications is adjusted to 30, i.e. n=30. This

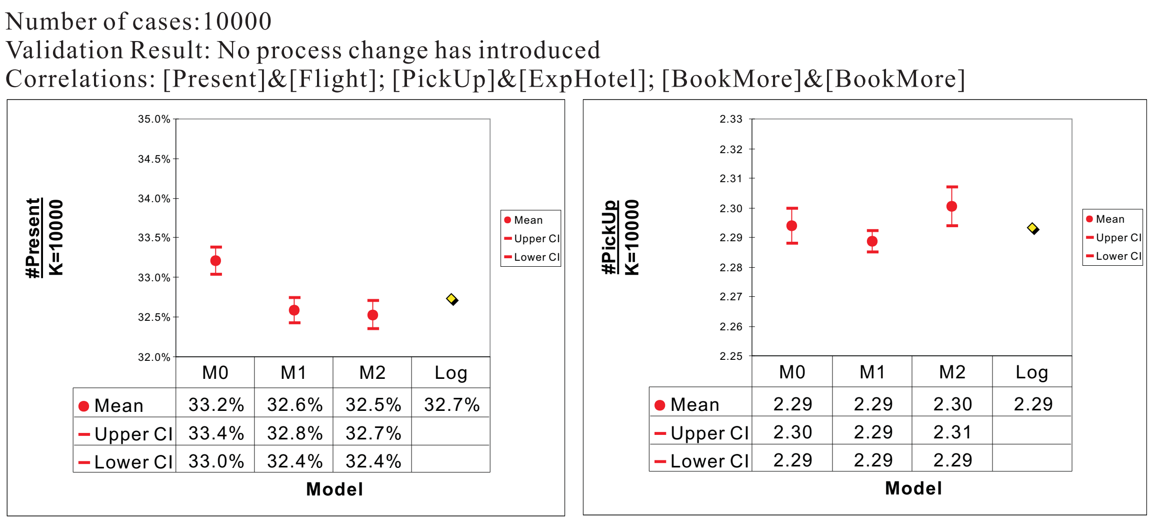
LFCT abstracted history	**Estimated p(t,h)**	**Mined p(t,h)**	**Relative Deviation ρ**
$ExpHotel$	52.5%	53.17%	1.3%
$MedHotel$	41%	39.11%	4.6%
$LowHotel$	44.15%	43.76%	0.8%
$history-independent$	46%	45.29%	1.5%

Table 5.2: Comparison between estimated and mined $p(t,h)$ values from a 1000-case log, where t="PickUp" and history is abstracted by the LFCT abstraction.

LFCT abstracted history	Estimated p(t,h)	Mined p(t,h)	Relative Deviation ρ
$ExpHotel$	52.5%	52.95%	0.9%
$MedHotel$	41%	41.36%	0.9%
$LowHotel$	44.15%	44.06%	0.2%
$history - independent$	46%	46.13%	0.3%

Table 5.3: Comparison between estimated and mined $p(t, h)$ values from a 10000-case log, where t="PickUp" and history is abstracted by the LFCT abstraction.

adjustment does not influence the results significantly. The new validation results with k=10000 and n=30 are shown in Figure 5.2. In Figure 5.2 (a), the first observation is that the increase of k reduced the size of the CIs, i.e. the results are more precise. With the reduced CIs, small gaps emerge between the CIs of indicator $\sharp Present$ for M_1 and M_2 and the CI for M_0. The reason is that the performance of the log (the diamond in Figure 5.2 (a)) selected for creating simulation models is out of the 95% estimate intervals of M_0. It is reasonable as the log is randomly selected. Whereas, the log performance is still within the estimate intervals of M_1 and M_2.


Number of cases: 10000
Validation Result: No process change has introduced
Correlations: [Present]&[Flight]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]

	M0	M1	M2	Log
● Mean	33.2%	32.6%	32.5%	32.7%
− Upper CI	33.4%	32.8%	32.7%	
− Lower CI	33.0%	32.4%	32.4%	

	M0	M1	M2	Log
● Mean	2.29	2.29	2.30	2.29
− Upper CI	2.30	2.29	2.31	
− Lower CI	2.29	2.29	2.29	

(a) TF abstraction: R([Present])=[Flight] (b) LFCT abstraction: R([PickUp])=[ExpHotel]

Figure 5.2: Validation result with 10000 cases and 30 replications without applying process changes. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. Note that we use a fraction scaling, which is $\frac{indicator}{k=10000}(\times 100\%)$.

In Figure 5.2 (b), statistical errors similarly result in fluctuations of the performance of M_1 and M_2 on the indicator $\sharp PickUp$. the log performance is still within the estimate intervals of M_1 and M_2. Therefore, we conclude that under the experiment setting, i.e. k=10000 and n=30, M_1 with correlation set 1 and M_2 both approximate M_0. Validation results for other correlation sets are placed in Appendix C.

5.2 Change Propagation Analysis with Process Changes

Since our two simulation models, M_1 and M_2, can approximate M_0 in absence of change. Now, we are ready to investigate the performances of M_1 and M_2 when process changes are injected into the process. In this section, we propose two scenarios where process changes are injected and we describe how these changes are injected. Finally, we present the experimental results.

5.2.1 Scenarios with Process changes

To investigate how decisions on *PickUp* and *Present* respond to process changes in M_0, M_1, and M_2, process changes should be introduced to the correlated clusters. Corresponding to the correlated clusters we determined in Table 5.1, we design two individual scenarios to apply changes as follows: the seasonal marketing research shows that (1) fewer clients prefer flights due to the economic crisis; (2) more clients prefer budget hotels due to the economic crisis.

Scenario	**Process Changes**	**Before Changes**	**After Changes**	**Estimated Effect**
Scenario(1)	$pFlight$	50%	10%	$\sharp Present \downarrow$,
	$pCruise$	50%	90%	$\sharp Pickup$ †
	$pExpHotel$	33.3%	10%	$\sharp Present \downarrow$,
Scenario(2)	$pMedHotel$	33.3%	25%	$\sharp Pickup \downarrow$
	$pLowHotel$	33.3%	65%	

Table 5.4: Scenarios designed for the change propagation analysis. $\downarrow$ stands for drops and † for remains.

In scenario (1), change is applied to the decision on $Flight$ and $Cruise$, i.e. the probability to book flights ($pFlight$) decreases. As fewer flights are taken, the total cost is lower than before such that fewer presents would be given in M_0 and M_1. The decision on the hotel is not changed such that the decision on a pickup service should not be affected in M_0 and M_1. In scenario (2), changes are applied to the decision on $ExpHotel$, $MedHotel$, and $LowHotel$, i.e. the probabilities to book luxury hotels ($pExpHotel$), middle-class hotels ($pMedHotel$), and budget hotels ($pLowHotel$) are changed. As fewer expensive and middle-class hotels are taken, the total cost is lower than before such that fewer presents would be given in M_0 and M_1. Meanwhile, fewer pickup services would be taken in M_0 and M_1 due to the increasing number of budget hotels booked. M_2 would not be affected during both scenarios. The changes on corresponding input parameters and the expected effects in M_0 and M_1 are summarized in Table 5.4.

5.2.2 Change Propagation Analysis Results

We simulate the proposed changes in Table 5.4 for different correlation sets in Table 5.1 with 10000 cases and 30 replications for M_0, M_1 and M_2 respectively. The remainder of this section describes the results.

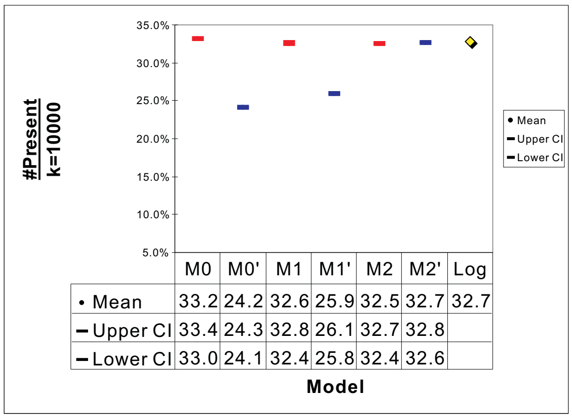
5.2.2.1 Correlation Set 1

In this section, we discuss the change propagation results over the LFCT abstraction correlation between [*PickUp*] and [*ExpHotel*], the TF abstraction correlation between the clusters [*BookMore*] and [*BookMore*], and the TF abstraction correlation between [*Present*] and [*Flight*].

In scenario (1), as estimated, due to fewer flights booked, the fraction of presents in M_0 indeed drops from 33.2% to 24.2% by 9% (see M_0 and M_0' in Figure 5.3 (a)). In M_1, as the process change is applied to the cluster [*Flight*], with which [*Present*] is strongly correlated, the fraction of presents in M_1 is affected, decreasing from 32.6% to 25.9% by around 6.7% (see M_1 and M_1' in Figure 5.3 (a)). Although the decrease in M_1 is not exactly same as M_0, the change is indeed propagated in M_1 approximately. Besides, as estimated, $\sharp$*PickUp* is not affected in the example process M_0 (see M_0' in Figure 5.3 (b)). For M_1, no effect emerges on the fraction of pickups as well. As the process change is applied to the cluster [*Flight*], with which [*PickUp*] is not strongly correlated, it is not surprising that the change on [*Flight*] is not propagated. The fraction of pickups and the fraction of presents in M_2 remain similar as expected.

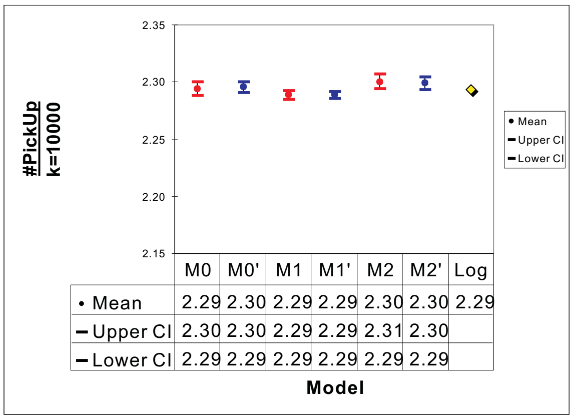
In scenario (2), as estimated, an increase in the booking of budget hotels, the fraction of presents in M_0 drops from 33.2% to 8.7% by around 24.5% (see M_0 and M_0' in Figure 5.4 (a)). In M_1, no effect emerges on the fraction of presents. It is not surprising because the process change is applied to the cluster [*ExpHotel*], while [*Present*] is strongly correlated with [*Flight*] under the TF abstraction. Meanwhile, as estimated, the fraction of pickups in M_0 decreases by 8% (see M_0 and M_0' in Figure 5.4 (b)). As the process change is applied to the cluster [*ExpHotel*], which [*PickUp*] is strongly correlated with under the LFCT abstraction, the fraction of pickups in M_1 is decreased by around 8.9% (see M_1 and M_1'

Scenario(1): Fewer clients prefer flights
Correlations: [Present]&[Flight]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]
Process Change: pFlight: 50% to 10%
pCruise: 50% to 90%
Estimated Effect: #Present drops and #PickUp remains



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	33.2	24.2	32.6	25.9	32.5	32.7	32.7
– Upper CI	33.4	24.3	32.8	26.1	32.7	32.8	
– Lower CI	33.0	24.1	32.4	25.8	32.4	32.6	

(a) TF abstraction: R([Present])=[Flight]



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	2.29	2.30	2.29	2.29	2.30	2.30	2.29
– Upper CI	2.30	2.30	2.29	2.29	2.31	2.30	
– Lower CI	2.29	2.29	2.29	2.29	2.29	2.29	

(b) LFCT abstraction: R([PickUp])=[ExpHotel]

Figure 5.3: Change propagation analysis results for scenario (1) with 10000 cases and 30 replications. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. M_0, M_1, and M_2 (M_0', M_1', and M_2') correspond to the results before (after) changes are applied.

in Figure 5.4 (b)). M_1 does propagate the changes successfully in this scenario while M_2 remains stable as expected.

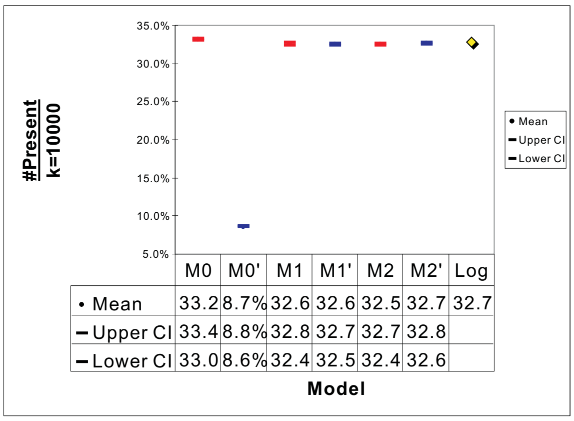
5.2.2.2 Correlation Set 2

In this section, we discuss the change propagation results over the LFCT abstraction correlation between $[PickUp]$ and $[ExpHotel]$, the TF abstraction correlation between the clusters $[BookMore]$ and $[BookMore]$, and the TF abstraction correlation between $[Present]$ and $[ExpHotel]$.

In scenario (1), as expected, the fraction of presents in M_0 drops by 9%. Unlike the behavior of M_1 in Figure 5.3 (a), M_1 in Figure 5.5 (a) fails to respond to the process change on the flights. It is because the process change is applied to the cluster $[Flight]$, while $[Present]$ is strongly correlated with $[ExpHotel]$ under the TF abstraction. M_2 is unable to capture the change for this correlation. Similar to the results in Figure 5.3 (b), the fraction of pickups remains similar in all models after the change is applied.

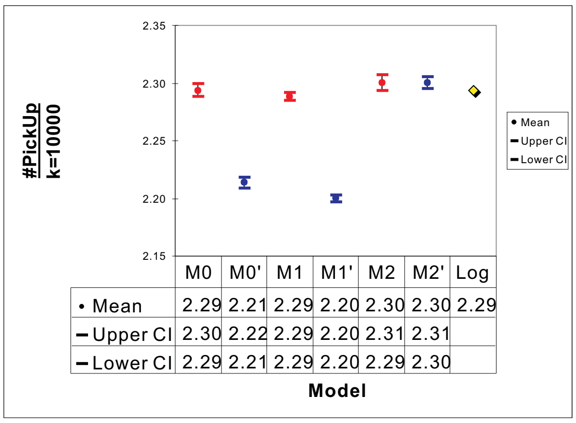
It is interesting to see the results for the scenario (2) in Figure 5.6. In this scenario, M_1

Scenario(2): More clients prefer budget hotels
Correlations: [Present]&[Flight]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]
Process Change: pExpHotel: 33% to 10%
pMedHotel: 33% to 25%
pLowHotel: 33% to 65%
Estimated Effect: #Present drops and #PickUp drops



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	33.2	8.7%	32.6	32.6	32.5	32.7	32.7
– Upper CI	33.4	8.8%	32.8	32.7	32.7	32.8	
– Lower CI	33.0	8.6%	32.4	32.5	32.4	32.6	

(a) TF abstraction: R([Present])=[Flight]



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	2.29	2.21	2.29	2.20	2.30	2.30	2.29
– Upper CI	2.30	2.22	2.29	2.20	2.31	2.31	
– Lower CI	2.29	2.21	2.29	2.20	2.29	2.30	

(b) LFCT abstraction: R([PickUp])=[ExpHotel]

Figure 5.4: Change propagation analysis results for scenario (2) with 10000 cases and 30 replications. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. M_0, M_1, and M_2 (M_0', M_1', and M_2') correspond to the results before (after) changes are applied.

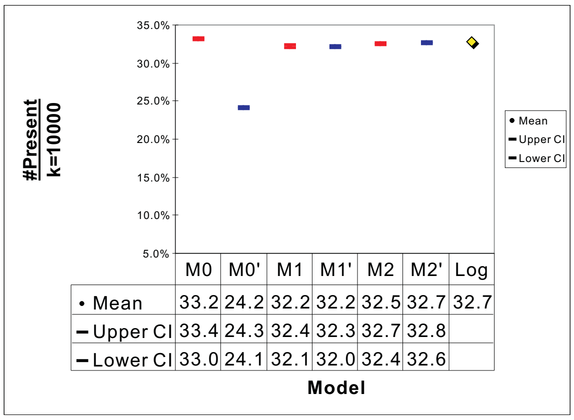
with the correlation set 2 propagates the change and yields a decrease of both indicators, indeed close to M_0. In M_1 (M_0), the decrease of the fraction of presents is 24% (24.5%) and the decrease of the fraction of pickups is 8.9% (8%). By comparison, M_2 fails to capture the process change. In this scenario and with this correlation setting, i.e. $R([Present]) = [ExpHotel]$, $R([PickUp]) = [ExpHotel]$, and $R([BookMore]) = [BookMore]$, it is clear that M_1 performs more precise than M_2.

5.2.2.3 Correlation Set 3

In this section, we discuss the change propagation results over the LFCT abstraction correlation between $[PickUp]$ and $[ExpHotel]$, the TF abstraction correlation between the clusters $[BookMore]$ and $[BookMore]$, and the TF abstraction correlation between $[Present]$ and $[BookMore]$.

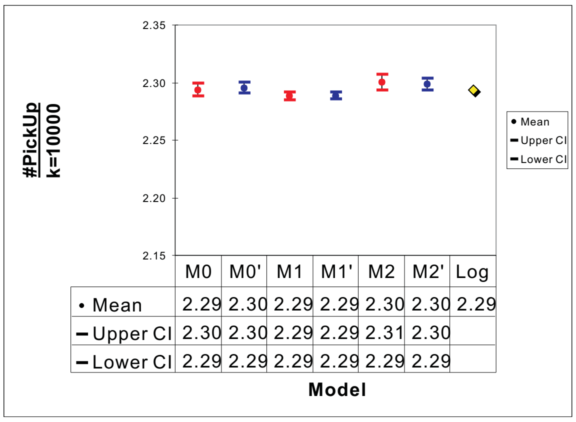
Unlike the last two correlation sets, with the correlation under the TF abstraction, M_1 fails to propagate any changes to transition *Present* in both scenarios (see Figure 5.7

Scenario(1): Fewer clients prefer flights
Correlations: [Present]&[ExpHotel]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]
Process Change: pFlight: 50% to 10%
pCruise: 50% to 90%
Estimated Effect: #Present drops and #PickUp remains



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	33.2	24.2	32.2	32.2	32.5	32.7	32.7
▬ Upper CI	33.4	24.3	32.4	32.3	32.7	32.8	
▬ Lower CI	33.0	24.1	32.1	32.0	32.4	32.6	

(a) TF abstraction: R([Present])=[ExpHotel]



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	2.29	2.30	2.29	2.29	2.30	2.30	2.29
▬ Upper CI	2.30	2.30	2.29	2.29	2.31	2.30	
▬ Lower CI	2.29	2.29	2.29	2.29	2.29	2.29	

(b) LFCT abstraction: R([PickUp])=[ExpHotel]

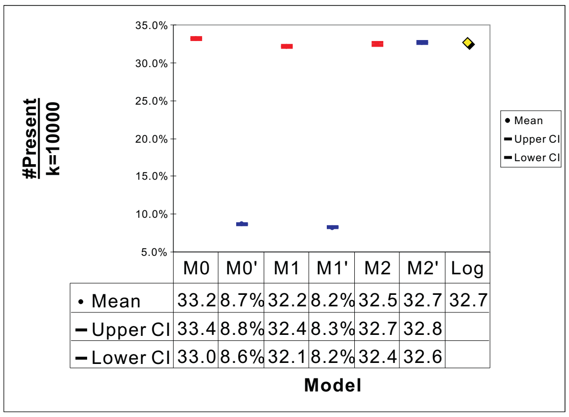
Figure 5.5: Change propagation analysis results for scenario (1) with 10000 cases and 30 replications. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. M_0, M_1, and M_2 (M_0', M_1', and M_2') correspond to the results before (after) changes are applied.

and Figure 5.8). It is reasonable that when process changes are applied to some decisions ($[Flight]$ and $[ExpHotel]$), with which the target decisions ($[Present]$ and $[PickUp]$) are not strongly correlated, no effect would be captured from the target decisions. In this situation, both M_1 and M_2 fail to give an accurate estimation of the performance indicators. However, when there is some change applied to the decision $[BookMore]$ in M_1, we believe that M_1 would be able to capture the propagated effect on the decision $[Present]$.

Previously, we compared the results across different models before and after applying changes. Now, let us take a look at the results from another perspective. To investigate the effect of changes on different correlation sets in some scenario, we compare the results across M_1 with different correlation sets before and after applying changes in some scenario. As M_2 does not capture any process change in both scenarios, this analysis is only conducted on M_1. Figure 5.9 and Figure 5.10 summarize the simulation results for M_1 with different correlation sets under two scenarios, where M_0 plays as a benchmark.

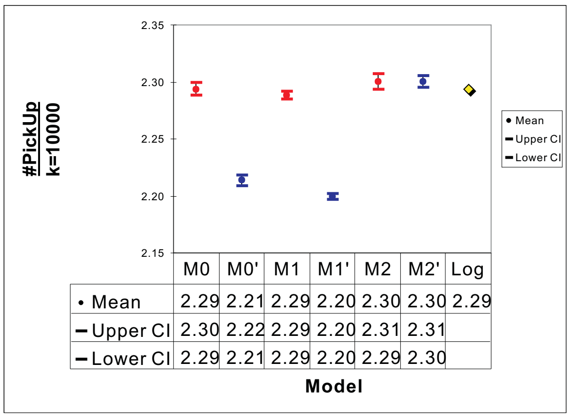
In scenario (1), the change is introduced to $[Flight]$. Only $M_1(R1)$, M_1 with correlation set 1, succeeds in capturing the change on the decision $[Present]$ (see Figure 5.9 (a) and (b)). However, in scenario (2), the change is introduced to $[ExpHotel]$. Only $M_1(R2)$,

Scenario(2): More clients prefer budget hotels
Correlations: [Present]&[ExpHotel]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]
Process Change: pExpHotel: 33% to 10%
pMedHotel: 33% to 25%
pLowHotel: 33% to 65%
Estimated Effect: #Present drops and #PickUp drops



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	33.2	8.7%	32.2	8.2%	32.5	32.7	32.7
– Upper CI	33.4	8.8%	32.4	8.3%	32.7	32.8	
– Lower CI	33.0	8.6%	32.1	8.2%	32.4	32.6	

(a) TF abstraction: R([Present])=[ExpHotel]



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	2.29	2.21	2.29	2.20	2.30	2.30	2.29
– Upper CI	2.30	2.22	2.29	2.20	2.31	2.31	
– Lower CI	2.29	2.21	2.29	2.20	2.29	2.30	

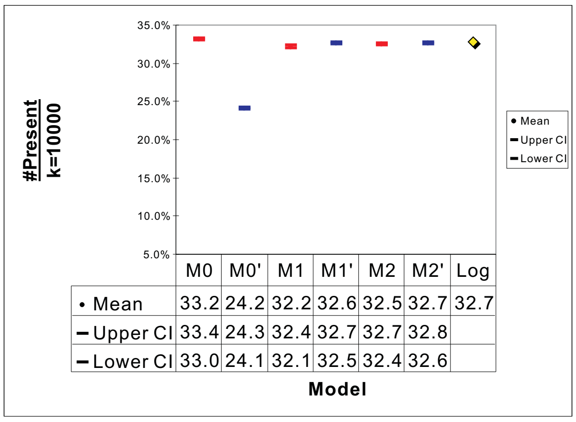
(b) LFCT abstraction: R([PickUp])=[ExpHotel]

Figure 5.6: Change propagation analysis results for scenario (2) with 10000 cases and 30 replications. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. M_0, M_1, and M_2 (M_0', M_1', and M_2') correspond to the results before (after) changes are applied.

M_1 with correlation set 2, succeeds in completely capturing the change on both decisions $[Present]$ and $[PickUp]$ (see Figure 5.10 (a) and (b)).

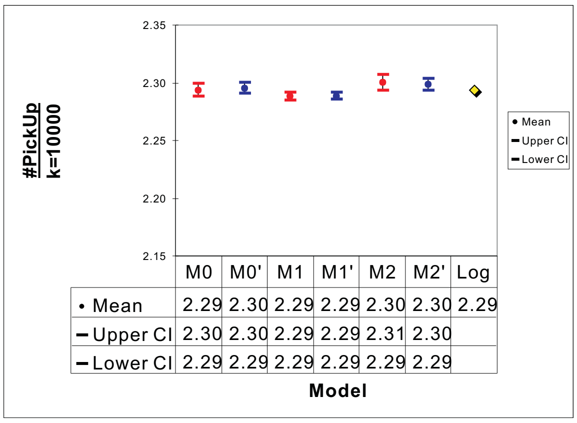
From the description above, we can see that the HDSPN is indeed able to propagate process changes in the process when the changes are applied to some correlation. However, the strongly correlated cluster is not always the best solution for selecting correlations, e.g. if the change affects a weaker correlated cluster and not the strongest one, then it might be better to use the weaker correlation. For instance (see Figure 5.10 $M_1(R1)$), the strongest correlation between $[Flight]$ and $[Present]$ under the TF abstraction fails to capture the change when the change is applied to another correlated cluster $[ExpHotel]$ (a weaker correlation). Thus, we say that the effect of the change propagation is local, depending on the correlation determined, the abstraction selected, and the change introduced. The locality of the change propagation motivates some other abstractions such as the extensive TF abstraction, which counts the frequency of every transition in the history without being restricted in one correlated cluster. By doing so, the factor of selecting one (strong or weak) correlation can probably be avoided. As it is outside the scope of this thesis, we do not investigate deeper. Further research is needed to address these issues.

Scenario(1): Fewer clients prefer flights
Correlations: [Present]&[BookMore]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]
Process Change: pFlight: 50% to 10%
pCruise: 50% to 90%
Estimated Effect: #Present drops and #PickUp remains



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	33.2	24.2	32.2	32.6	32.5	32.7	32.7
▬ Upper CI	33.4	24.3	32.4	32.7	32.7	32.8	
▬ Lower CI	33.0	24.1	32.1	32.5	32.4	32.6	

(a) TF abstraction: R([Present])=[BookMore]

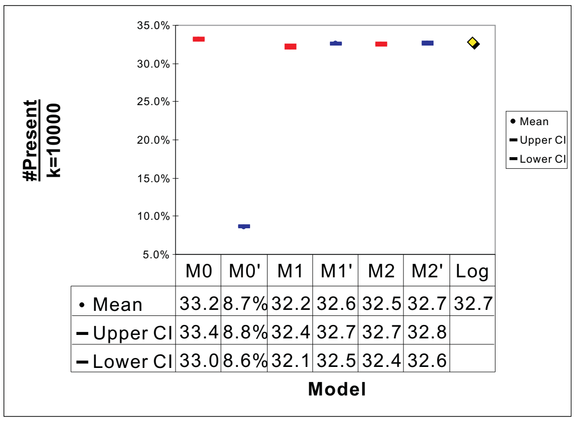


	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	2.29	2.30	2.29	2.29	2.30	2.30	2.29
▬ Upper CI	2.30	2.30	2.29	2.29	2.31	2.30	
▬ Lower CI	2.29	2.29	2.29	2.29	2.29	2.29	

(b) LFCT abstraction: R([PickUp])=[ExpHotel]

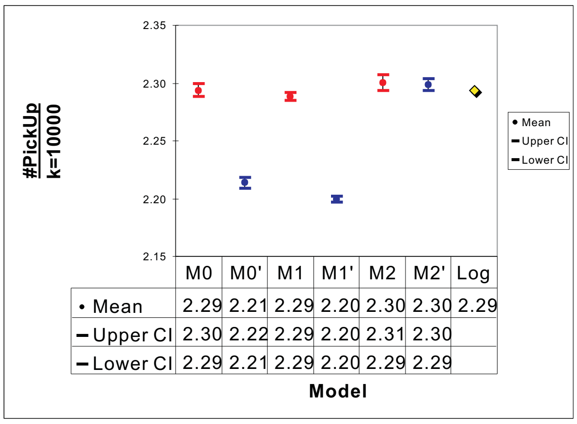
Figure 5.7: Change propagation analysis results for scenario (1) with 10000 cases and 30 replications. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. M_0, M_1, and M_2 (M_0', M_1', and M_2') correspond to the results before (after) changes are applied.

Scenario(2): More clients prefer budget hotels
Correlations: [Present]&[BookMore]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]
Process Change: pExpHotel: 33% to 10%
pMedHotel: 33% to 25%
pLowHotel: 33% to 65%
Estimated Effect: #Present drops and #PickUp drops



	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	33.2	8.7%	32.2	32.6	32.5	32.7	32.7
− Upper CI	33.4	8.8%	32.4	32.7	32.7	32.8	
− Lower CI	33.0	8.6%	32.1	32.5	32.4	32.6	

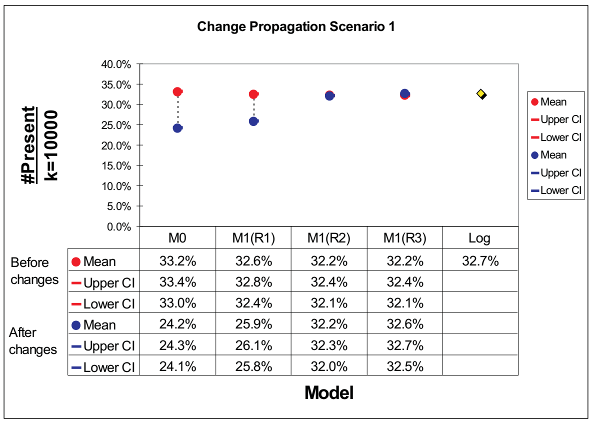
(a) TF abstraction: R([Present])=[BookMore]



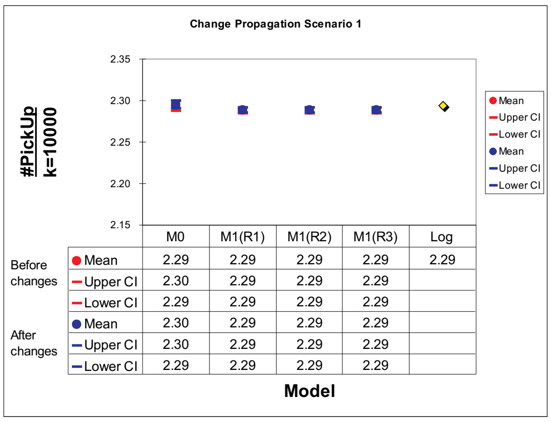
	M0	M0'	M1	M1'	M2	M2'	Log
• Mean	2.29	2.21	2.29	2.20	2.30	2.30	2.29
− Upper CI	2.30	2.22	2.29	2.20	2.31	2.30	
− Lower CI	2.29	2.21	2.29	2.20	2.29	2.29	

(b) LFCT abstraction: R([PickUp])=[ExpHotel]

Figure 5.8: Change propagation analysis results for scenario (2) with 10000 cases and 30 replications. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. M_0, M_1, and M_2 (M_0', M_1', and M_2') correspond to the results before (after) changes are applied.

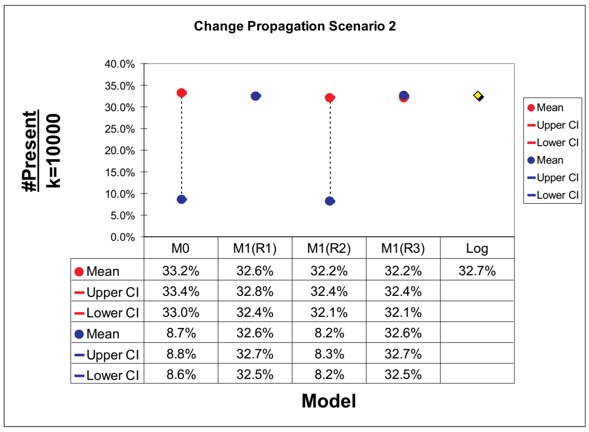


		M0	M1(R1)	M1(R2)	M1(R3)	Log
Before changes	● Mean	33.2%	32.6%	32.2%	32.2%	32.7%
	Upper CI	33.4%	32.8%	32.4%	32.4%	
	Lower CI	33.0%	32.4%	32.1%	32.1%	
After changes	● Mean	24.2%	25.9%	32.2%	32.6%	
	Upper CI	24.3%	26.1%	32.3%	32.7%	
	Lower CI	24.1%	25.8%	32.0%	32.5%	

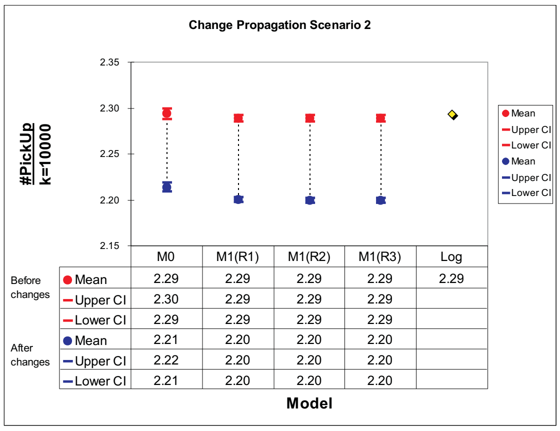


		M0	M1(R1)	M1(R2)	M1(R3)	Log
Before changes	● Mean	2.29	2.29	2.29	2.29	2.29
	Upper CI	2.30	2.29	2.29	2.29	
	Lower CI	2.29	2.29	2.29	2.29	
After changes	● Mean	2.30	2.29	2.29	2.29	
	Upper CI	2.30	2.29	2.29	2.29	
	Lower CI	2.29	2.29	2.29	2.29	

Figure 5.9: Performance of M_1 with different correlation sets for scenario (1) with 10000 cases and 30 replications. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. The figures in red (in blue) refer to the results before (after) changes are applied. $M_1(R_1)$ stands for the M_1 with the correlation set 1, so as for $M_1(R_2)$ and $M_1(R_3)$.



	M0	M1(R1)	M1(R2)	M1(R3)	Log
● Mean	33.2%	32.6%	32.2%	32.2%	32.7%
Upper CI	33.4%	32.8%	32.4%	32.4%	
Lower CI	33.0%	32.4%	32.1%	32.1%	
● Mean	8.7%	32.6%	8.2%	32.6%	
Upper CI	8.8%	32.7%	8.3%	32.7%	
Lower CI	8.6%	32.5%	8.2%	32.5%	



		M0	M1(R1)	M1(R2)	M1(R3)	Log
Before changes	● Mean	2.29	2.29	2.29	2.29	2.29
	Upper CI	2.30	2.29	2.29	2.29	
	Lower CI	2.29	2.29	2.29	2.29	
After changes	● Mean	2.21	2.20	2.20	2.20	
	Upper CI	2.22	2.20	2.20	2.20	
	Lower CI	2.21	2.20	2.20	2.20	

Figure 5.10: Performance of M_1 with different correlation sets for scenario (2) with 10000 cases and 30 replications. The diamond refers to the performance mined from the log that is used to create simulation models. The left figure (a) depicts the performances of three models on indicator $\sharp Present$. The right figure (b) depicts the performances of three models on indicator $\sharp PickUp$. The figures in red (in blue) refer to the results before (after) changes are applied. $M_1(R_1)$ stands for the M_1 with the correlation set 1, so as for $M_1(R_2)$ and $M_1(R_3)$.

Chapter 6

Conclusions

The intention of this thesis is to verify that HDSPNs are indeed able to propagate process changes through the process by simulations. This chapter concludes what we have achieved. Some issues that this thesis does not cover are discussed in the remainder.

6.1 Achievements

To obtain a log and the corresponding control flow suitable for the change propagation analysis, we built an example process, shown in Figure 4.5, with data dependencies as Colored Petri nets. To build a HDSPN, we introduced an algorithm with two abstraction functions (the TF abstraction and the LFCT abstraction) to discover correlations. In addition, the algorithm was implemented as a plug-in in the ProM framework. An approach about how to calculate the transition probabilities from the log has been proposed. Having the mined correlation and the calculated transition probability values, we constructed the HDSPN which mimicked the example process.

Having the HDSPN model, we proposed a set of experiments to verify the change propagation of the HDSPN. To compare the HDSPN framework with an existing logged simulation model framework, Rozinat et al.'s extracted simulation model was introduced into the simulation experiments. The experiments started with the validation without introducing changes. Through the validation, we intended to assess if the HDSPN and Rozinat et al.'s simulation model approximate the behavior of the example process without introducing changes by comparing the frequencies of the decisions (denoted by D_1). These decisions were determined to be correlated with decisions made before (denoted by D_2). Without change injection, both HDSPNs framework and Rozinat et al.'s simulation framework closely approximate the example process. However, the small-size log, which is used for creating the HDSPN, could probably give unreliable history-dependent transition probabilities for HDSPN framework.

Afterwards, we conducted the change propagation analysis by injecting process changes on decisions D_2. By observing the frequencies of decisions D_1, we assessed if the HDSPN captured the process changes approximately as the example process did. The simulation experimental results showed that when process changes are injected, to some extent (depending on the correlation determined, the abstraction selected, and the change introduced), HDSPNs are able to propagate process changes in the process, while Rozinat et al.'s simulation framework is indeed incapable to capture the changes as the correlations are missing.

The most important results of this thesis are as follows. (1) From the experimental results, we find out that when the change is applied to a weaker correlation, the strongest correlation performs so poorly in propagating the change that no change would be captured. Therefore, we say that to assess the change propagation, it is not always a good option to select the strongest correlation. The selection of correlations should also consider the change, e.g. where the change is applied. (2) Correlations are dependent on the abstractions used. A strongest correlation under the TF abstraction is probably not the strongest one under the LFCT abstraction. Therefore, the change propagation of HDSPNs depends on the change where to apply, correlation selection, and the abstraction used.

One practical application of our work is to evaluate the process recommenders. Process recommenders offer support to end users by suggesting which alternative could be done first, based on historical information from an event log. Currently, several recommendation techniques have been proposed. Evaluations of these techniques are necessary before they can be used in a practical setting. Simulation models are used to simulate the effect of recommendations on the running process and to compare different recommendation techniques. As process changes are normally brought into the process by the process recommenders to adjust the process executions, the simulation models to evaluate the recommendations are required to be able to capture the change through the processes. Thus, compared with the existing framework, HDSPNs could be a better simulation framework for evaluating process recommenders .

6.2 Future Work

Although we have achieved the goal of this thesis, several issues, which this thesis does not cover, are required to investigate further in the future.

First of all, as we stated one cluster is likely correlated with multiple clusters. In this thesis, we only consider the strongly correlated cluster. However, we did not use any scientific mechanism to determine the strength of the correlations. Instead, we determined the strongest correlation according to the domain expert knowledge of the process. Further work should focus on data mining techniques or regression analysis to detect the strength of the correlations.

During observing the log, we noticed that the log observation matrix mined under the TF abstraction is sparse due to the existence of iterations in the process. The more iterations are incorporated in the process, sparser the log observation matrix would be. A sparse matrix may result in the unreliable test results of the chi-square test. For this problem, machine learning techniques such as matrix clustering may help.

As we also argued, the correlation discovery algorithm is sensitive to the log size. If the number of cases in the log, which is used for constructing HDSPNs, is not sufficient, the mined correlations and transition probabilities are unreliable. In any case, small log-sizes give less reliable results. Of course the same number of probabilities will be mined, but the estimations are based on smaller samples, or even different estimation techniques could be used if observations are missing. Large-size logs are recommended for mining correlations and transition probabilities. For smaller logs, the results are more unreliable.

The last issue is disposed from the simulation experimental results. The experimental results showed that the HDSPN is indeed able to propagate process changes in the process when the changes are applied to some correlation. However, the experimental results implied that the strongly correlated cluster is not always the best solution for selecting correlations, e.g. if the change affects a weaker correlated cluster and not the strongest one, then it might be better to use the weaker correlation. Thus, the effect of the change propagation is local, depending on the correlation determined, the abstraction selected, and the change introduced. The locality of the change propagation motivates some alternatives: (1) "global" or "trace" abstraction, which is not restricted in one cluster; and (2) combination of abstractions (strong and weak(er) abstractions).

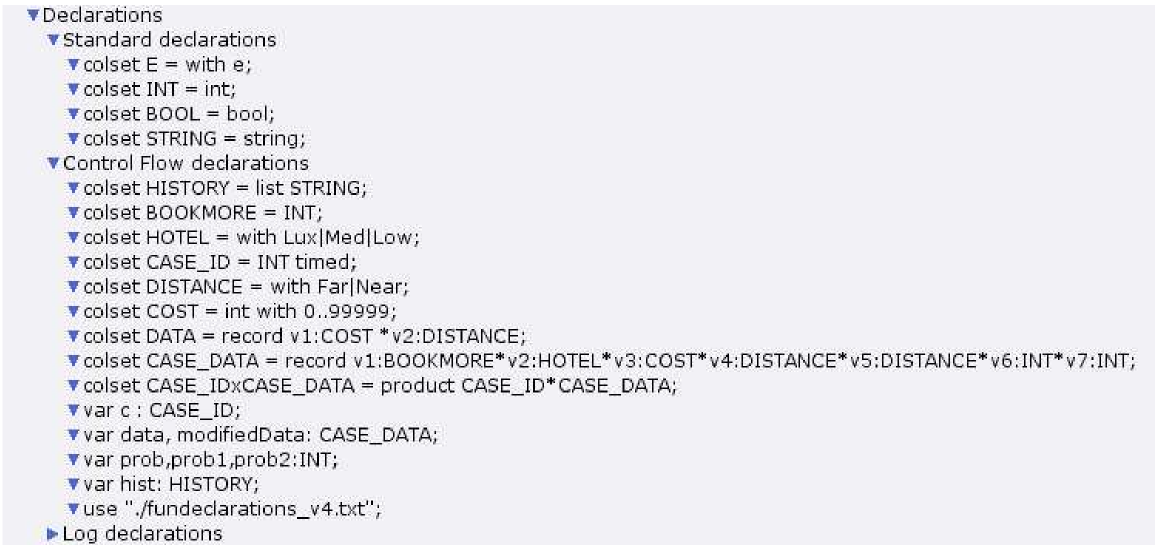
By applying the alternative (1), the factor of selecting (strong or weak) correlations can probably be avoided and the quality of abstractions could be improved. However, as the history is less abstracted by high-quality abstractions, we will normally find out that we have not enough data in our log to make any reliable estimations. More estimations from the structure of the net are used such that the reliability of our estimations could be harmed. The balance between the reliability and the quality of the abstractions is an interesting topic for the future work. By applying the alternative (2), the factor of selecting (strong or weak) correlations can probably be avoided. However, the contingency matrix (row identifiers) and the $p(t, h)$ estimations should be reconsidered probably.

Appendix A

ML Implementations in CPN Tools

A.1 ML Implementations in M_0

Part of the declarations for the ML implementations in M_0 are captured in Figure A.1. The user defined ML functions are implemented as shown in Figure A.2. Monitors required to produce the log and measure the performance are shown in Figure A.3.

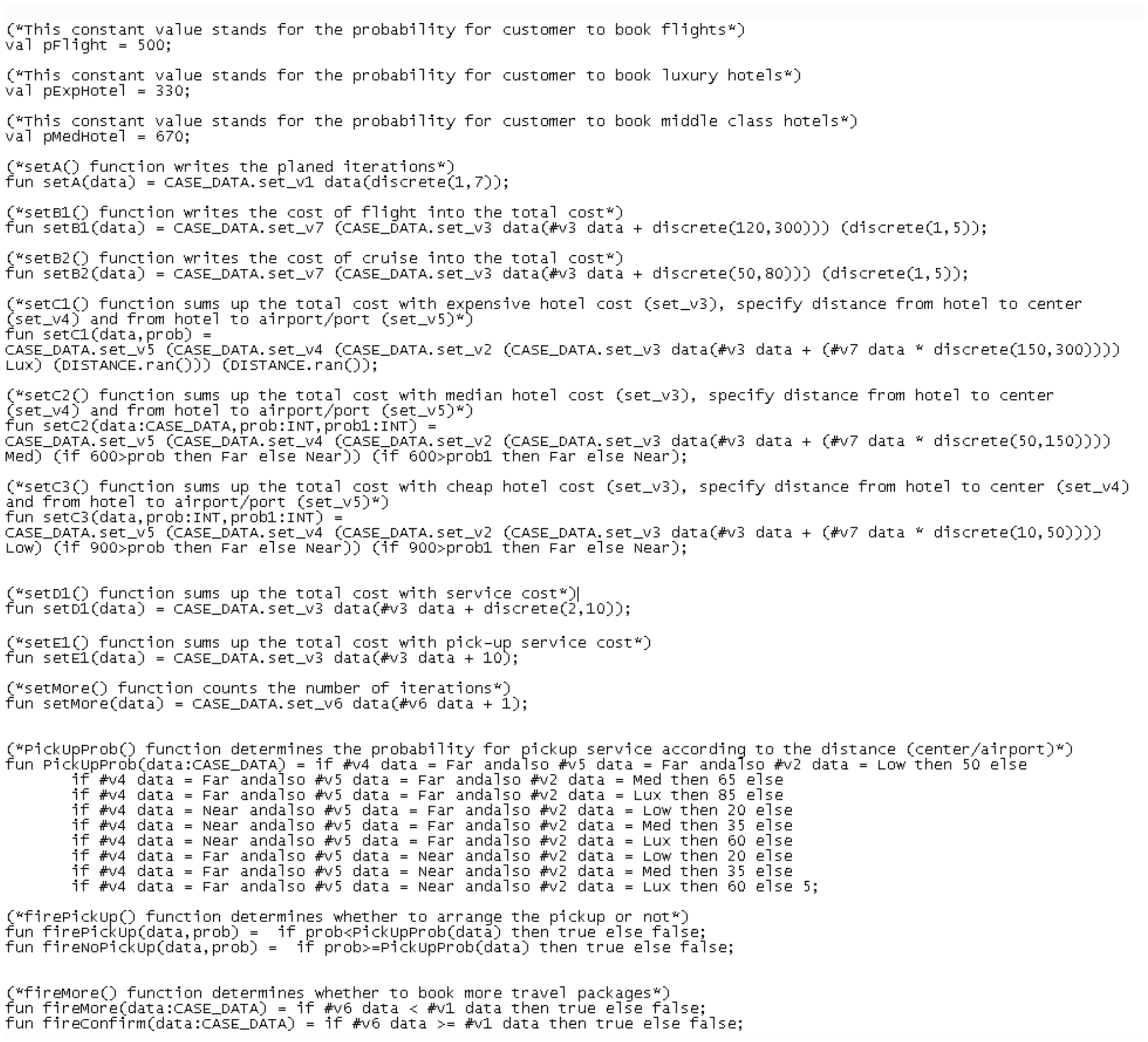


```
▼Declarations
  ▼Standard declarations
    ▼colset E = with e;
    ▼colset INT = int;
    ▼colset BOOL = bool;
    ▼colset STRING = string;
  ▼Control Flow declarations
    ▼colset HISTORY = list STRING;
    ▼colset BOOKMORE = INT;
    ▼colset HOTEL = with Lux|Med|Low;
    ▼colset CASE_ID = INT timed;
    ▼colset DISTANCE = with Far|Near;
    ▼colset COST = int with 0..99999;
    ▼colset DATA = record v1:COST *v2:DISTANCE;
    ▼colset CASE_DATA = record v1:BOOKMORE*v2:HOTEL*v3:COST*v4:DISTANCE*v5:DISTANCE*v6:INT*v7:INT;
    ▼colset CASE_IDxCASE_DATA = product CASE_ID*CASE_DATA;
    ▼var c : CASE_ID;
    ▼var data, modifiedData: CASE_DATA;
    ▼var prob,prob1,prob2:INT;
    ▼var hist: HISTORY;
    ▼use "./fundeclarations_v4.txt";
  ▶Log declarations
```

Figure A.1: CPN declarations in M_0. Note that the declarations related with the log generation are not shown. They are explained in [15] in detail.

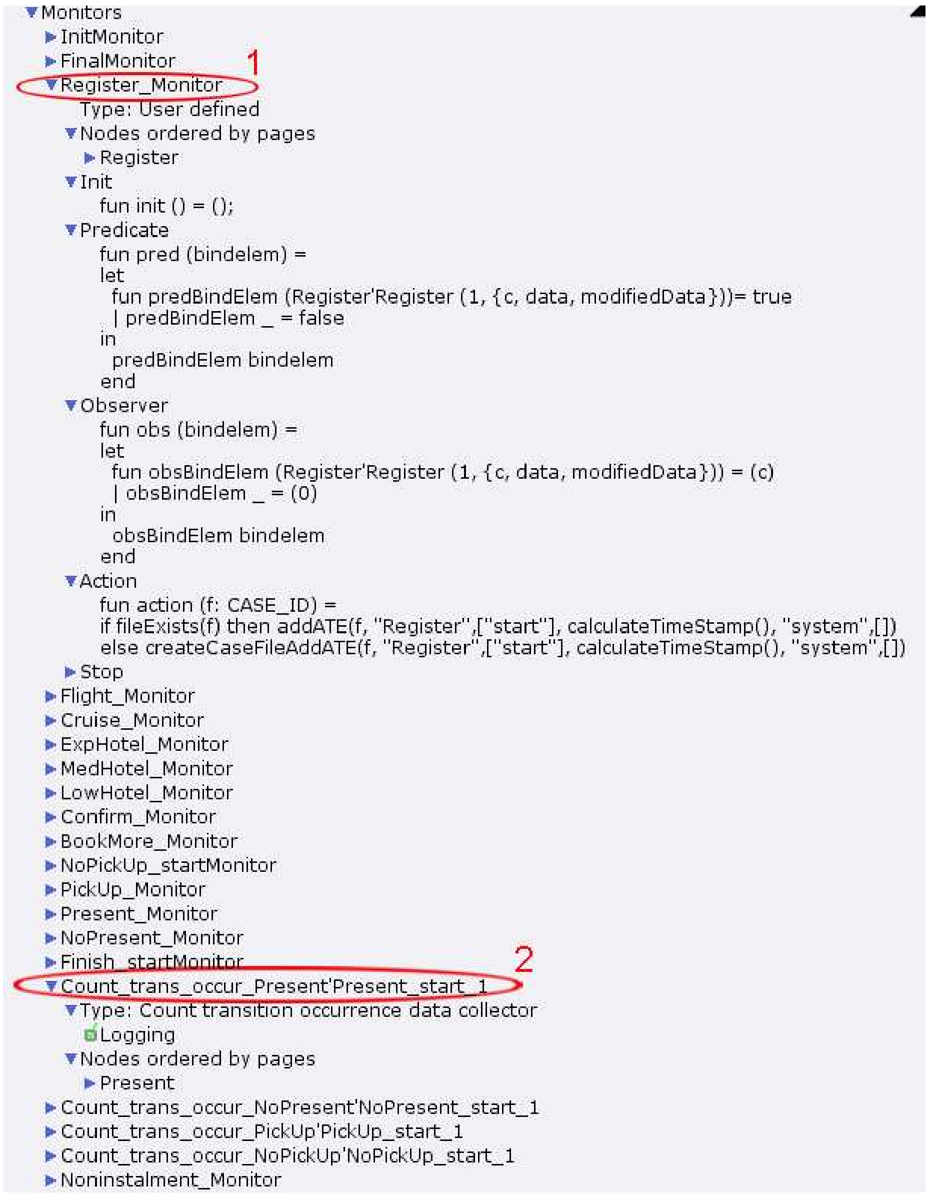
A.2 ML Implementations in M_1

Part of the declarations for the ML implementations in M_1 are captured in Figure A.4. The user defined ML functions are implemented as shown in Figure A.5. Monitors required to measure the performance are same as in M_0 (see Figure A.3).



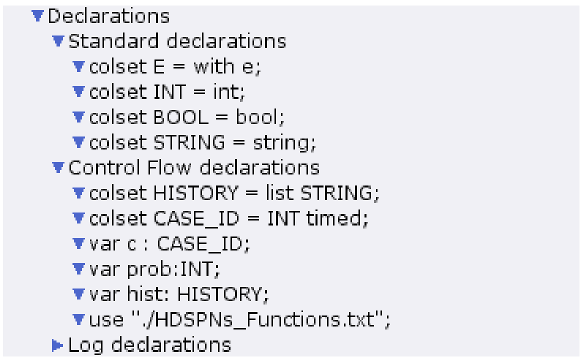
```
(*This constant value stands for the probability for customer to book flights*)
val pFlight = 500;

(*This constant value stands for the probability for customer to book luxury hotels*)
val pExpHotel = 330;

(*This constant value stands for the probability for customer to book middle class hotels*)
val pMedHotel = 670;

(*setA() function writes the planed iterations*)
fun setA(data) = CASE_DATA.set_v1 data(discrete(1,7));

(*setB1() function writes the cost of flight into the total cost*)
fun setB1(data) = CASE_DATA.set_v7 (CASE_DATA.set_v3 data(#v3 data + discrete(120,300))) (discrete(1,5));

(*setB2() function writes the cost of cruise into the total cost*)
fun setB2(data) = CASE_DATA.set_v7 (CASE_DATA.set_v3 data(#v3 data + discrete(50,80))) (discrete(1,5));

(*setC1() function sums up the total cost with expensive hotel cost (set_v3), specify distance from hotel to center
(set_v4) and from hotel to airport/port (set_v5)*)
fun setC1(data,prob) =
CASE_DATA.set_v5 (CASE_DATA.set_v4 (CASE_DATA.set_v2 (CASE_DATA.set_v3 data(#v3 data + (#v7 data * discrete(150,300))))
Lux) (DISTANCE.ran())) (DISTANCE.ran());

(*setC2() function sums up the total cost with median hotel cost (set_v3), specify distance from hotel to center
(set_v4) and from hotel to airport/port (set_v5)*)
fun setC2(data:CASE_DATA,prob:INT,prob1:INT) =
CASE_DATA.set_v5 (CASE_DATA.set_v4 (CASE_DATA.set_v2 (CASE_DATA.set_v3 data(#v3 data + (#v7 data * discrete(50,150))))
Med) (if 600>prob then Far else Near)) (if 600>prob1 then Far else Near);

(*setC3() function sums up the total cost with cheap hotel cost (set_v3), specify distance from hotel to center (set_v4)
and from hotel to airport/port (set_v5)*)
fun setC3(data,prob:INT,prob1:INT) =
CASE_DATA.set_v5 (CASE_DATA.set_v4 (CASE_DATA.set_v2 (CASE_DATA.set_v3 data(#v3 data + (#v7 data * discrete(10,50))))
Low) (if 900>prob then Far else Near)) (if 900>prob1 then Far else Near);


(*setD1() function sums up the total cost with service cost*)
fun setD1(data) = CASE_DATA.set_v3 data(#v3 data + discrete(2,10));

(*setE1() function sums up the total cost with pick-up service cost*)
fun setE1(data) = CASE_DATA.set_v3 data(#v3 data + 10);

(*setMore() function counts the number of iterations*)
fun setMore(data) = CASE_DATA.set_v6 data(#v6 data + 1);


(*PickUpProb() function determines the probability for pickup service according to the distance (center/airport)*)
fun PickUpProb(data:CASE_DATA) = if #v4 data = Far andalso #v5 data = Far andalso #v2 data = Low then 50 else
       if #v4 data = Far andalso #v5 data = Far andalso #v2 data = Med then 65 else
       if #v4 data = Far andalso #v5 data = Far andalso #v2 data = Lux then 85 else
       if #v4 data = Near andalso #v5 data = Far andalso #v2 data = Low then 20 else
       if #v4 data = Near andalso #v5 data = Far andalso #v2 data = Med then 35 else
       if #v4 data = Near andalso #v5 data = Far andalso #v2 data = Lux then 60 else
       if #v4 data = Far andalso #v5 data = Near andalso #v2 data = Low then 20 else
       if #v4 data = Far andalso #v5 data = Near andalso #v2 data = Med then 35 else
       if #v4 data = Far andalso #v5 data = Near andalso #v2 data = Lux then 60 else 5;

(*firePickUp() function determines whether to arrange the pickup or not*)
fun firePickUp(data,prob) =  if prob<PickUpProb(data) then true else false;
fun fireNoPickUp(data,prob) =  if prob>=PickUpProb(data) then true else false;


(*fireMore() function determines whether to book more travel packages*)
fun fireMore(data:CASE_DATA) = if #v6 data < #v1 data then true else false;
fun fireConfirm(data:CASE_DATA) = if #v6 data >= #v1 data then true else false;
```

Figure A.2: User defined ML functions in M_0.



```
▼Monitors
  ▶InitMonitor
  ▶FinalMonitor
  ▼Register_Monitor
    Type: User defined
    ▼Nodes ordered by pages
      ▶Register
    ▼Init
      fun init () = ();
    ▼Predicate
      fun pred (bindelem) =
      let
        fun predBindElem (Register'Register (1, {c, data, modifiedData}))= true
        | predBindElem _ = false
      in
        predBindElem bindelem
      end
    ▼Observer
      fun obs (bindelem) =
      let
        fun obsBindElem (Register'Register (1, {c, data, modifiedData})) = (c)
        | obsBindElem _ = (0)
      in
        obsBindElem bindelem
      end
    ▼Action
      fun action (f: CASE_ID) =
      if fileExists(f) then addATE(f, "Register",["start"], calculateTimeStamp(), "system",[])
      else createCaseFileAddATE(f, "Register",["start"], calculateTimeStamp(), "system",[])
    ▶Stop
  ▶Flight_Monitor
  ▶Cruise_Monitor
  ▶ExpHotel_Monitor
  ▶MedHotel_Monitor
  ▶LowHotel_Monitor
  ▶Confirm_Monitor
  ▶BookMore_Monitor
  ▶NoPickUp_startMonitor
  ▶PickUp_Monitor
  ▶Present_Monitor
  ▶NoPresent_Monitor
  ▶Finish_startMonitor
  ▼Count_trans_occur_Present'Present_start_1
    ▼Type: Count transition occurrence data collector
      Logging
    ▼Nodes ordered by pages
      ▶Present
  ▶Count_trans_occur_NoPresent'NoPresent_start_1
  ▶Count_trans_occur_PickUp'PickUp_start_1
  ▶Count_trans_occur_NoPickUp'NoPickUp_start_1
  ▶Noninstalment_Monitor
```

Figure A.3: CPN monitors in M_0. There are two types of monitors, one for generating the log and the other for measuring the performance. Red circle 1 highlights the monitor defined for logging transition *Register*. Red circle 2 highlights the monitor defined for measuring the occurrences of transition *Present* in the simulations.



```
▼Declarations
  ▼Standard declarations
    ▼colset E = with e;
    ▼colset INT = int;
    ▼colset BOOL = bool;
    ▼colset STRING = string;
  ▼Control Flow declarations
    ▼colset HISTORY = list STRING;
    ▼colset CASE_ID = INT timed;
    ▼var c : CASE_ID;
    ▼var prob:INT;
    ▼var hist: HISTORY;
    ▼use "./HDSPNs_Functions.txt";
  ▶Log declarations
```

Figure A.4: CPN declarations in M_1. Note that the declarations related with the log generation are not shown. They are explained in [15] in detail.

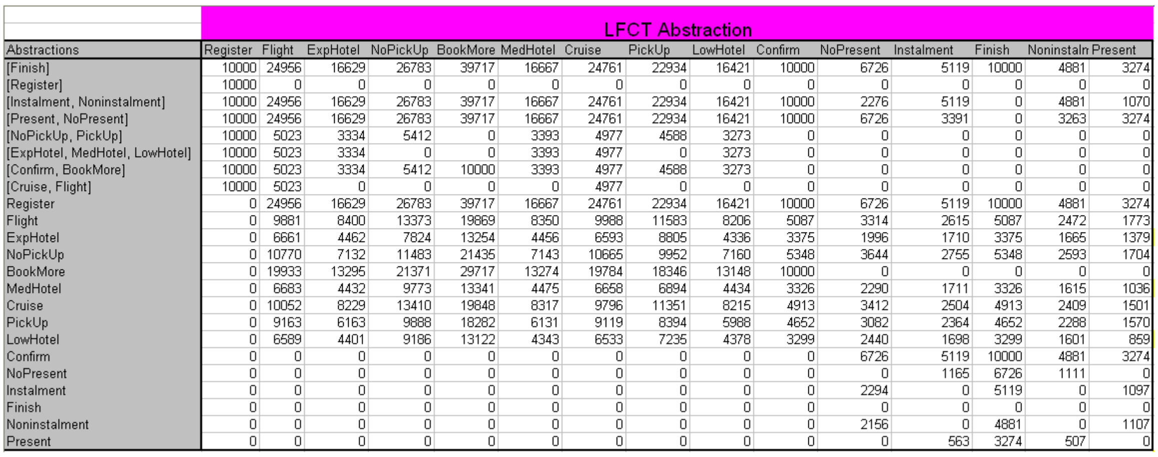
```
(*These are the probabilities for customer to book flights/ExpHotel/MedHotel*)
val pTrans = 500;
val pExpHotel = 330;
val pMedHotel = 670;


(*These functions (values) are defined to abstract history by LFCT abstraction function*)
val cluster =  ["ExpHotel","MedHotel","LowHotel"];
fun mem [] a = false | mem (x::xs) a = a=x orelse mem xs a;
fun getLFCT([]) = "null" | getLFCT(h::hist) = if mem cluster h then h else getLFCT(hist);


(*These functions (values) are defined to abstract history by TF abstraction function*)
fun count(s, [])= [] | count(s, h::hist) = if h=s then h::count(s,hist) else count(s,hist);
fun getFrequency(s, [])= 0 | getFrequency(s,h::hist) = length (count(s, h::hist));
fun getTF([]) = {flight=0,cruise=0} | getTF(h::hist) = {flight=getFrequency("Flight",h::hist), cruise=getFrequency("Cruise",h::hist)};
fun getTF1([]) = {confirm=0,bookmore=0} | getTF1(h::hist) = {confirm=getFrequency("Confirm",h::hist), bookmore=getFrequency("BookMore",h::hist)};
fun getTF2([]) = {exphotel=0,medhotel=0,lowhotel=0} | getTF2(h::hist) = {exphotel=getFrequency("ExpHotel",h::hist),
                    medhotel=getFrequency("MedHotel",h::hist), lowhotel=getFrequency("LowHotel",h::hist)};


(*This function select a p(t,h) value for [PickUp] according to the last fired transition in [ExpHotel]*)
fun LFCTprob([]) = 500 | LFCTprob(h::hist) = if getLFCT(h::hist) = "ExpHotel" then 529 else
                                             if getLFCT(h::hist) = "MedHotel" then 414 else
                                             if getLFCT(h::hist) = "LowHotel" then 441 else 461;


(*This function select a p(t,h) value for [Present] according to the TF abstraction value on the history of [Flight]*)
fun TFprob([]) = 500 | TFprob(h::hist) = if getTF(h::hist) = {flight=2,cruise=0} then 0 else
                                         if getTF(h::hist) = {flight=3,cruise=0} then 6 else
                                         if getTF(h::hist) = {flight=4,cruise=0} then 82 else
                                         if getTF(h::hist) = {flight=5,cruise=0
                                         if getTF(h::hist) = {flight=6,cruise=0
                                         if getTF(h::hist) = {flight=7,cruise=0
                                         if getTF(h::hist) = {flight=8,cruise=0
                                         if getTF(h::hist) = {flight=1,cruise=1
                                         if getTF(h::hist) = {flight=2,cruise=1
                                         if getTF(h::hist) = {flight=3,cruise=1
                                         if getTF(h::hist) = {flight=4,cruise=1
                                         if getTF(h::hist) = {flight=5,cruise=1
                                         if getTF(h::hist) = {flight=6,cruise=1
                                         if getTF(h::hist) = {flight=7,cruise=1} then 1000 else
                                         if getTF(h::hist) = {flight=0,cruise=2} then 0 else
                                         if getTF(h::hist) = {flight=1,cruise=2} then 7 else
                                         if getTF(h::hist) = {flight=2,cruise=2} then 90 else
                                         if getTF(h::hist) = {flight=3,cruise=2} then 246 else
                                         if getTF(h::hist) = {flight=4,cruise=2} then 543 else
                                         if getTF(h::hist) = {flight=5,cruise=2} then 769 else
                                         if getTF(h::hist) = {flight=6,cruise=2} then 949 else
                                         if getTF(h::hist) = {flight=0,cruise=3} then 0 else
                                         if getTF(h::hist) = {flight=1,cruise=3} then 54 else
                                         if getTF(h::hist) = {flight=2,cruise=3} then 224 else
                                         if getTF(h::hist) = {flight=3,cruise=3} then 455 else
                                         if getTF(h::hist) = {flight=4,cruise=3} then 683 else
                                         if getTF(h::hist) = {flight=5,cruise=3} then 852 else
                                         if getTF(h::hist) = {flight=0,cruise=4} then 57 else
                                         if getTF(h::hist) = {flight=1,cruise=4} then 166 else
                                         if getTF(h::hist) = {flight=2,cruise=4} then 393 else
                                         if getTF(h::hist) = {flight=3,cruise=4} then 634 else
                                         if getTF(h::hist) = {flight=4,cruise=4} then 841 else
                                         if getTF(h::hist) = {flight=0,cruise=5} then 77 else
                                         if getTF(h::hist) = {flight=1,cruise=5} then 402 else
                                         if getTF(h::hist) = {flight=2,cruise=5} then 601 else
                                         if getTF(h::hist) = {flight=3,cruise=5} then 817 else
                                         if getTF(h::hist) = {flight=0,cruise=6} then 235 else
                                         if getTF(h::hist) = {flight=1,cruise=6} then 532 else
                                         if getTF(h::hist) = {flight=2,cruise=6} then 729 else
                                         if getTF(h::hist) = {flight=0,cruise=7} then 750 else
                                         if getTF(h::hist) = {flight=1,cruise=7} then 725 else
                                         if getTF(h::hist) = {flight=0,cruise=8} then 667 else 327;

(*This function select a p(t,h) value for [Present] according to the TF abstraction value on the history of [Flight]*)
fun TFprob2([]) = 500 | TFprob2(h::hist) = if #exphotel (getTF2(h::hist))=0 then HotelProb0(h::hist) else
                                           if #exphotel (getTF2(h::hist))=1 then HotelProb1(h::hist) else
                                           if #exphotel (getTF2(h::hist))=2 then HotelProb2(h::hist) else
                                           if #exphotel (getTF2(h::hist))=3 then HotelProb3(h::hist) else
                                           if #exphotel (getTF2(h::hist))=4 then HotelProb4(h::hist) else
                                           if #exphotel (getTF2(h::hist))=5 then HotelProb5(h::hist) else
                                           if #exphotel (getTF2(h::hist))>=6 then HotelProb6(h::hist) else 327;


(*This function select a p(t,h) value for [Present] according to the TF abstraction value on the history of [BookMore]*)
fun TFprob1([]) = 500 | TFprob1(h::hist) = if getTF1(h::hist) = {confirm=1,bookmore=1} then 0 else
                                         if getTF1(h::hist) = {confirm=1,bookmore=2} then 9 else
                                         if getTF1(h::hist) = {confirm=1,bookmore=3} then 87 else
                                         if getTF1(h::hist) = {confirm=1,bookmore=4} then 243 else
                                         if getTF1(h::hist) = {confirm=1,bookmore=5} then 478 else
                                         if getTF1(h::hist) = {confirm=1,bookmore=6} then 668 else
                                         if getTF1(h::hist) = {confirm=1,bookmore=7} then 837 else 327;


(*This function select a p(t,h) value for [BookMore] according to the TF abstraction value on the history of [BookMore]*)
fun TFprobBook([]) = 500 | TFprobBook(h::hist) = if getTF1(h::hist) = {confirm=0,bookmore=0} then 1000 else
                                         if getTF1(h::hist) = {confirm=0,bookmore=1} then 855 else
                                         if getTF1(h::hist) = {confirm=0,bookmore=2} then 831 else
                                         if getTF1(h::hist) = {confirm=0,bookmore=3} then 794 else
                                         if getTF1(h::hist) = {confirm=0,bookmore=4} then 750 else
                                         if getTF1(h::hist) = {confirm=0,bookmore=5} then 665 else
                                         if getTF1(h::hist) = {confirm=0,bookmore=6} then 463 else
                                         if getTF1(h::hist) = {confirm=0,bookmore=7} then 0 else 799;


(*This function returns true if the random value is smaller than the selected p(t,h) value for "BookMore"*)
fun fireBookMore(prob,hist)= if prob<TFprobBook(hist) then true else false;

(*This function returns true if the random value is not smaller than the selected p(t,h) value for "Confirm"*)
fun fireConfirm(prob,hist)= if prob>=TFprobBook(hist)9:12 PM 8/18/2009 then true else false;


(*This function returns true if the random value is smaller than the selected p(t,h) value for "PickUp"*)
fun firePickUp(prob,hist)= if prob<LFCTprob(hist) then true else false;

(*This function returns true if the random value is not smaller than the selected p(t,h) value for "NoPickUp"*)
fun fireNoPickUp(prob,hist)= if prob>=LFCTprob(hist) then true else false;


(*This function returns true if the random value is smaller than the selected p(t,h) value for "Present"*)
fun firePresent(prob,hist) = if prob<TFprob2(hist) then true else false;


(*This function returns true if the random value is not smaller than the selected p(t,h) value for "NoPresent"*)
fun fireNoPresent(prob,hist) = if prob>=TFprob2(hist) then true else false;
```

Figure A.5: User defined ML functions in M_1. Note that in this implementation cluster [Present] is set to be correlated with cluster [ExpHotel].

Appendix B

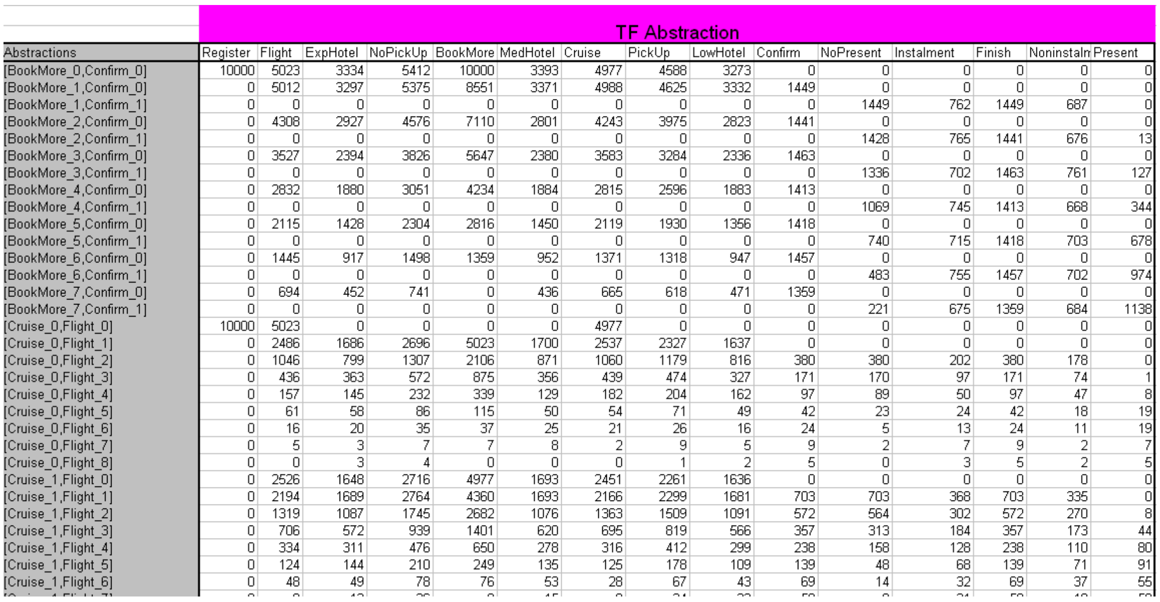
Correlation Mining Results

Here we present the log observation matrices mined by our "cluster correlation miner". The matrices are used for calculating $p^{lfct}(t,h)$ and $p^{tf}(t,h)$ respectively. Due to the size of the matrix, the complete log observation matrix for the LFCT abstraction are shown while for the TF abstraction only part of the matrix is shown.



	LFCT Abstraction														
Abstractions	Register	Flight	ExpHotel	NoPickUp	BookMore	MedHotel	Cruise	PickUp	LowHotel	Confirm	NoPresent	Instalment	Finish	Noninstalm	Present
[Finish]	10000	24956	16629	26783	39717	16667	24761	22934	16421	10000	6726	5119	10000	4881	3274
[Register]	10000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
[Instalment, Noninstalment]	10000	24956	16629	26783	39717	16667	24761	22934	16421	10000	2276	5119	0	4881	1070
[Present, NoPresent]	10000	24956	16629	26783	39717	16667	24761	22934	16421	10000	6726	3391	0	3263	3274
[NoPickUp, PickUp]	10000	5023	3334	5412	0	3393	4977	4588	3273	0	0	0	0	0	0
[ExpHotel, MedHotel, LowHotel]	10000	5023	3334	0	0	3393	4977	0	3273	0	0	0	0	0	0
[Confirm, BookMore]	10000	5023	3334	5412	10000	3393	4977	4588	3273	0	0	0	0	0	0
[Cruise, Flight]	10000	5023	0	0	0	0	4977	0	0	0	0	0	0	0	0
Register	0	24956	16629	26783	39717	16667	24761	22934	16421	10000	6726	5119	10000	4881	3274
Flight	0	9881	8400	13373	19869	8350	9988	11583	8206	5087	3314	2615	5087	2472	1773
ExpHotel	0	6661	4462	7824	13254	4456	6593	8805	4336	3375	1996	1710	3375	1665	1379
NoPickUp	0	10770	7132	11483	21435	7143	10665	9952	7160	5348	3644	2755	5348	2593	1704
BookMore	0	19933	13295	21371	29717	13274	19784	18346	13148	10000	0	0	0	0	0
MedHotel	0	6683	4432	9773	13341	4475	6658	6894	4434	3326	2290	1711	3326	1615	1036
Cruise	0	10052	8229	13410	19848	8317	9796	11351	8215	4913	3412	2504	4913	2409	1501
PickUp	0	9163	6163	9888	18282	6131	9119	8394	5988	4652	3082	2364	4652	2288	1570
LowHotel	0	6589	4401	9186	13122	4343	6533	7235	4378	3299	2440	1698	3299	1601	859
Confirm	0	0	0	0	0	0	0	0	0	0	6726	5119	10000	4881	3274
NoPresent	0	0	0	0	0	0	0	0	0	0	0	1165	6726	1111	0
Instalment	0	0	0	0	0	0	0	0	0	0	2294	0	5119	0	1097
Finish	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Noninstalment	0	0	0	0	0	0	0	0	0	0	2156	0	4881	0	1107
Present	0	0	0	0	0	0	0	0	0	0	0	563	3274	507	0

Figure B.1: Log observation matrix for the LFCT abstraction. Abstraction values with [ ] refer to an undefined transition. Abstraction values without [ ] refer to the last fired transition name.



	TF Abstraction													
Abstractions	Register	Flight	ExpHotel	NoPickUp	BookMore	MedHotel	Cruise	PickUp	LowHotel	Confirm	NoPresent	Instalment	Finish	Noninstalm	Present
[BookMore_0,Confirm_0]	10000	5023	3334	5412	10000	3393	4977	4588	3273	0	0	0	0	0	0
[BookMore_1,Confirm_0]	0	5012	3297	5375	8551	3371	4988	4625	3332	1449	0	0	0	0	0
[BookMore_1,Confirm_1]	0	0	0	0	0	0	0	0	0	0	1449	762	1449	687	0
[BookMore_2,Confirm_0]	0	4308	2927	4576	7110	2801	4243	3975	2823	1441	0	0	0	0	0
[BookMore_2,Confirm_1]	0	0	0	0	0	0	0	0	0	0	1428	765	1441	676	13
[BookMore_3,Confirm_0]	0	3527	2394	3826	5647	2380	3583	3284	2336	1463	0	0	0	0	0
[BookMore_3,Confirm_1]	0	0	0	0	0	0	0	0	0	0	1336	702	1463	761	127
[BookMore_4,Confirm_0]	0	2832	1880	3051	4234	1884	2815	2596	1883	1413	0	0	0	0	0
[BookMore_4,Confirm_1]	0	0	0	0	0	0	0	0	0	0	1069	745	1413	668	344
[BookMore_5,Confirm_0]	0	2115	1428	2304	2816	1450	2119	1930	1356	1418	0	0	0	0	0
[BookMore_5,Confirm_1]	0	0	0	0	0	0	0	0	0	0	740	715	1418	703	678
[BookMore_6,Confirm_0]	0	1445	917	1498	1359	952	1371	1318	947	1457	0	0	0	0	0
[BookMore_6,Confirm_1]	0	0	0	0	0	0	0	0	0	0	483	755	1457	702	974
[BookMore_7,Confirm_0]	0	694	452	741	0	436	665	618	471	1359	0	0	0	0	0
[BookMore_7,Confirm_1]	0	0	0	0	0	0	0	0	0	0	221	675	1359	684	1138
[Cruise_0,Flight_0]	10000	5023	0	0	0	0	4977	0	0	0	0	0	0	0	0
[Cruise_0,Flight_1]	0	2486	1686	2696	5023	1700	2537	2327	1637	0	0	0	0	0	0
[Cruise_0,Flight_2]	0	1046	799	1307	2106	871	1060	1179	816	380	380	202	380	178	0
[Cruise_0,Flight_3]	0	436	363	572	875	356	439	474	327	171	170	97	171	74	1
[Cruise_0,Flight_4]	0	157	145	232	339	129	182	204	162	97	89	50	97	47	8
[Cruise_0,Flight_5]	0	61	58	86	115	50	54	71	49	42	23	24	42	18	19
[Cruise_0,Flight_6]	0	16	20	35	37	25	21	26	16	24	5	13	24	11	19
[Cruise_0,Flight_7]	0	5	3	7	7	8	2	9	5	9	2	7	9	2	7
[Cruise_0,Flight_8]	0	0	3	4	0	0	0	1	2	5	0	3	5	2	5
[Cruise_1,Flight_0]	0	2526	1648	2716	4977	1693	2451	2261	1636	0	0	0	0	0	0
[Cruise_1,Flight_1]	0	2194	1689	2764	4360	1693	2166	2299	1681	703	703	368	703	335	0
[Cruise_1,Flight_2]	0	1319	1087	1745	2682	1076	1363	1509	1091	572	564	302	572	270	8
[Cruise_1,Flight_3]	0	706	572	939	1401	620	695	819	566	357	313	184	357	173	44
[Cruise_1,Flight_4]	0	334	311	476	650	278	316	412	299	238	158	128	238	110	80
[Cruise_1,Flight_5]	0	124	144	210	249	135	125	178	109	139	48	68	139	71	91
[Cruise_1,Flight_6]	0	48	49	78	76	53	28	67	43	69	14	32	69	37	55

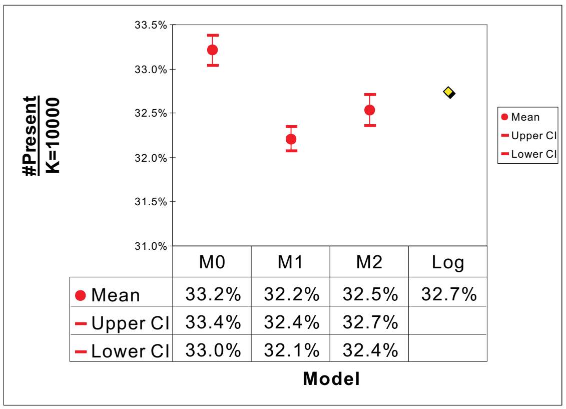
Figure B.2: Part of the log observation matrix for the TF abstraction. The string in the [ ] refers to the cluster transition, and the number followed by refers to the transition's frequency.

Appendix C

Simulation Results

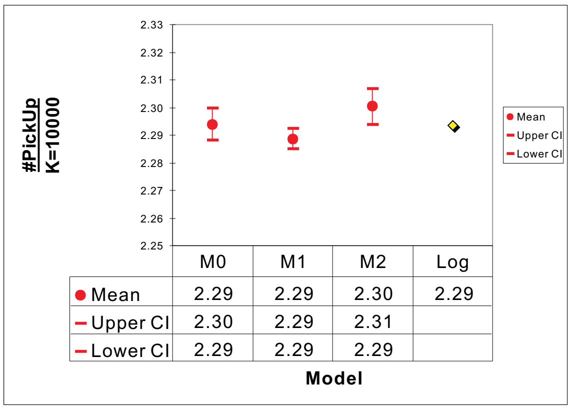
C.1 Validation Results w.r.t Correlation Set 2

Number of cases:10000
Validation Result: No process change has introduced
Correlations: [Present]&[ExpHotel]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]



	M0	M1	M2	Log
● Mean	33.2%	32.2%	32.5%	32.7%
– Upper CI	33.4%	32.4%	32.7%	
– Lower CI	33.0%	32.1%	32.4%	

(a) TF abstraction: R([Present])=[ExpHotel]



	M0	M1	M2	Log
● Mean	2.29	2.29	2.30	2.29
– Upper CI	2.30	2.29	2.31	
– Lower CI	2.29	2.29	2.29	

(b) LFCT abstraction: R([PickUp])=[ExpHotel]

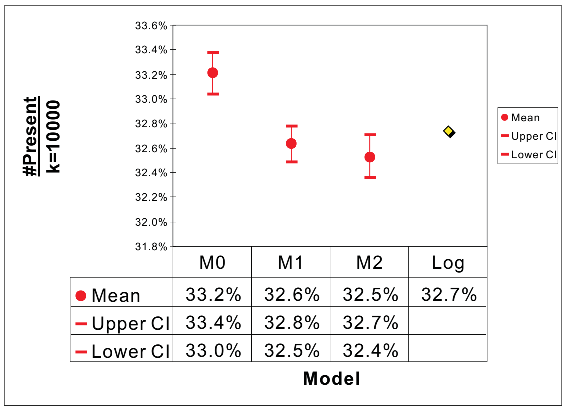
Figure C.1: Validation result with 10000 cases and 30 replications without applying process changes. The diamond refers to the performances mined from the log used to integrate simulation models. The left depicts the performances of three models on indicator $\sharp Present$. The right depicts the performances of three models on indicator $\sharp PickUp$. Note that we use a fraction scaling, which is $\frac{indicator}{k=10000}(\times 100\%)$.

Here, we present the validation results before introducing changes over the LFCT abstraction correlation between $[PickUp]$ and $[ExpHotel]$, the TF abstraction correlation between $[Present]$ and $[ExpHotel]$, and another TF abstraction correlation between $[BookMore]$ and $[BookMore]$.

The first observation is that the increase of k reduced the size of the CIs, i.e. the results are more precise. With the reduced CIs, small gaps emerge between the CIs of indicator $\sharp Present$ for M_1 and M_2 and the CI for M_0. The reason is still caused by the performance of the log (the diamond in Figure C.1 (a)) selected for creating simulation models. Due to the statistical errors, the true performance (log) is out of the 95% CI for M_1. In Figure C.1 (b). However, the relative error is round 1.5% which is negligible according to Rule 5.1. Statistical errors similarly result in negligible fluctuations of the performance of M_1 and M_2 on the indicator $\sharp PickUp$. Therefore, we conclude that M_1 with correlation set 2 and M_2 both approximate M_0.

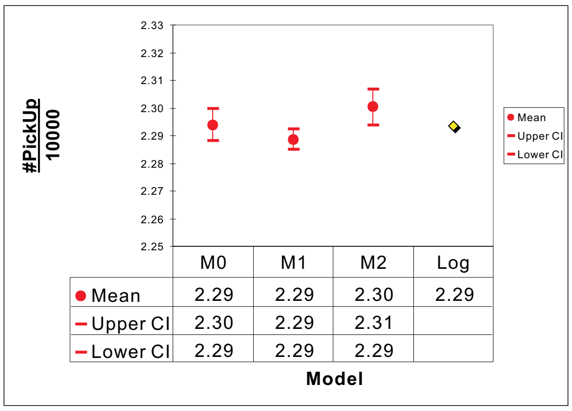
C.2 Validation Results w.r.t Correlation Set 3

Number of cases:10000
Validation Result: No process change has introduced
Correlations: [Present]&[BookMore]; [PickUp]&[ExpHotel]; [BookMore]&[BookMore]



	M0	M1	M2	Log
● Mean	33.2%	32.6%	32.5%	32.7%
– Upper CI	33.4%	32.8%	32.7%	
– Lower CI	33.0%	32.5%	32.4%	

(a) TF abstraction: R([Present])=[BookMore]



	M0	M1	M2	Log
● Mean	2.29	2.29	2.30	2.29
– Upper CI	2.30	2.29	2.31	
– Lower CI	2.29	2.29	2.29	

(b) LFCT abstraction: R([PickUp])=[ExpHotel]

Figure C.2: Validation result with 10000 cases and 30 replications without applying process changes. The diamond refers to the performances mined from the log used to integrate simulation models. The left depicts the performances of three models on indicator $\sharp Present$. The right depicts the performances of three models on indicator $\sharp PickUp$. Note that we use a percentage scaling, which is $\frac{indicator}{k=10000}(\times 100\%)$.

Here, we present the validation results before introducing changes over the LFCT abstraction correlation between $[PickUp]$ and $[ExpHotel]$, the TF abstraction correlation between $[Present]$ and $[BookMore]$, and another TF abstraction correlation between $[BookMore]$ and $[BookMore]$.

The first observation is that the increase of k reduced the size of the CIs, i.e. the results are more precise. With the reduced CIs, small gaps emerge between the CIs of indicator $\sharp Present$ for M_1 and M_2 and the CI for M_0. The reason is still caused by the performance of the log (the diamond in Figure C.2 (a)) selected for creating simulation models. Whereas, the log performance is still within the estimate intervals of M_1 and M_2. In Figure C.2 (b), statistical errors similarly result in negligible fluctuations of the performance of M_1 and M_2 on the indicator $\sharp PickUp$. Therefore, we conclude that M_1 with correlation set 3 and M_2 both approximate M_0.

Appendix D

Cluster Correlation Miner Manual

Here, we introduce two ways to execute the "cluster correlation miner" in ProM framework. One is to execute step by step manually. The other is to define a macro to set up the working environment. In other words, we can chain the required (input) objects together and then execute the miner.

D.1 Execution without Macro

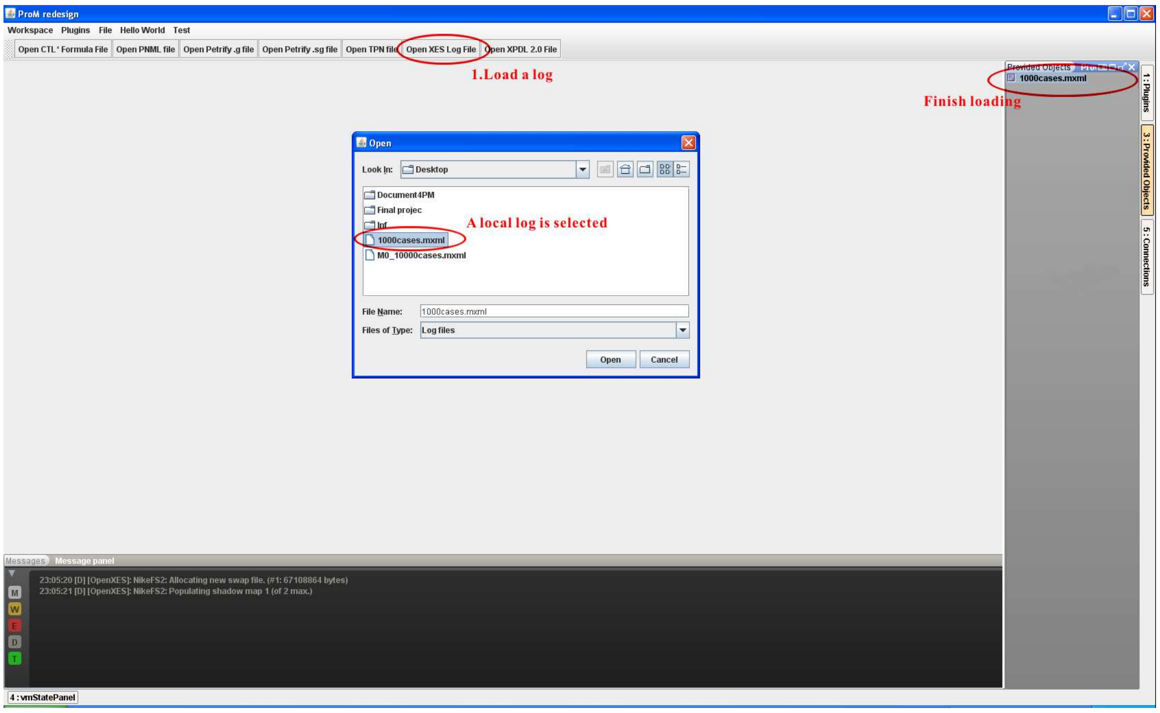
To execute the "cluster correlation miner" in the ProM framework step by step manually, the following procedures should be followed.

1. Load the log from which you need to derive the log observation matrices and to mine the correlations. The loaded log is placed in the "provided objects" list (see in Figure D.1).

2. Mine the cluster set of the net from the log. Right click the log in the "provided objects" list $\Longrightarrow$ Select available plugins $\Longrightarrow$ Select "Cluster/Decision Miner" (Default settins). The mined cluster set of the net would appear in the "provided objects" list. It is possible to view the mined cluster set by right-clicking the cluster set and selecting "show".

3. Choose the abstraction(s) to derive the log observation matrices and the correlations. Select both objects, i.e. the log and the cluster set, $\Longrightarrow$ Select available plugins $\Longrightarrow$ Select "Cluster Correlation Miner (Select options to use)" (see in Figure D.2). The pop-up GUI provides all available abstractions (multi-choices are allowed).

4. The log observation matrices for selected abstraction(s) can be viewed by right-clicking the provided object "Log Observations" and "show" (see in Figure D.3). The correlation results can be viewed in the same way. Note that the log observation matrix



```
23:05:20 [D] [OpenXES]: NikeFS2: Allocating new swap file. (#1: 67108864 bytes)
23:05:21 [D] [OpenXES]: NikeFS2: Populating shadow map 1 (of 2 max.)
```

Figure D.1: Load the log.

shown on the GUI might be only part of the actual log observation matrix. Due to the scale of the GUI, when the matrix is too big, not the whole matrix is presented. "Export" feature can help to export the selected log observation matrix to txt/csv file.

D.2 Execution with Macro

To define a macro to set up the working environment, the following procedures should be followed.

1. Load the log from which you need to derive the log observation matrices and to mine the correlations. The loaded log would be placed in the "provided objects" list (see in Figure D.1).

2. Define a macro to set the miner working environment. The "cluster correlation miner" requires a log and a cluster set as input. We chain the log with the "cluster correlation miner" and also with the "Cluster/Decision Miner". The output of the "Cluster/Decision Miner", i.e. the cluster set, is then chained with the "cluster correlation miner". The chained objects are shown in Figure D.4. To execute the macro, do not forget to specify

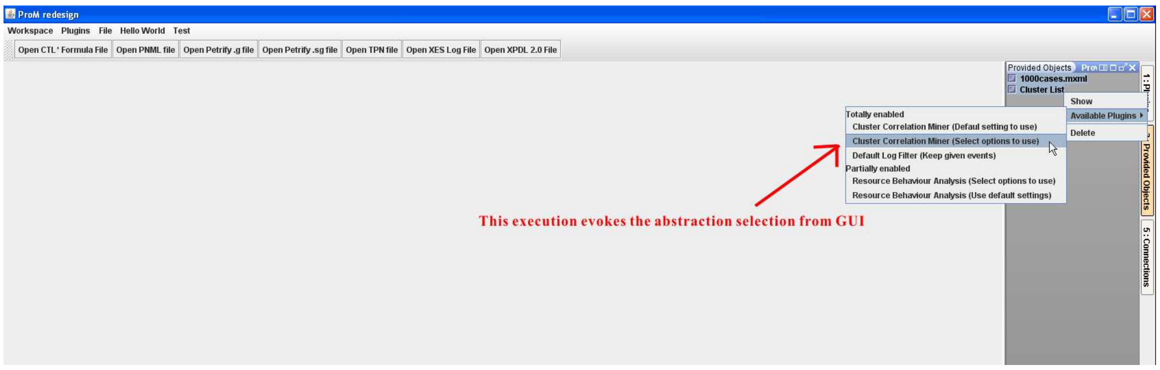
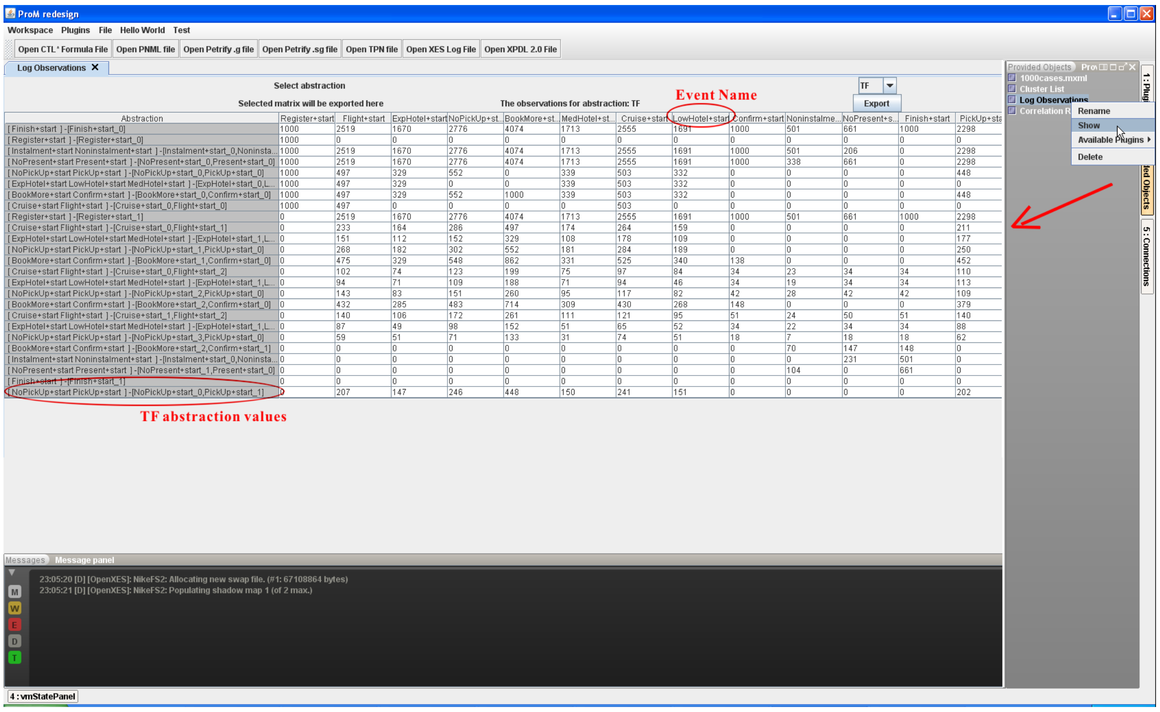

ProM redesign
Workspace Plugins File Hello World Test
Open CTL* Formula File
Open PNML file
Open Petrify .g file
Open Petrify .sg file
Open TPN file
Open XES Log File
Open XPDL 2.0 File
Provided Objects
1000cases.mxml
Cluster List
Show
Available Plugins
Delete
Totally enabled
Cluster Correlation Miner (Defaul setting to use)
Cluster Correlation Miner (Select options to use)
Default Log Filter (Keep given events)
Partially enabled
Resource Behaviour Analysis (Select options to use)
Resource Behaviour Analysis (Use default settings)
This execution evokes the abstraction selection from GUI
1 : Pl
Provided Objects
5 : Connections

Figure D.2: Choose the abstraction(s) to derive the log observation matrices and the correlations.

which execution type is preferred (GUI or the default setting).

To derive the log observation matrices and correlation results, the following procedures are same as we introduced in Appendix D.1 (from step 3 to step 4).



Selected matrix will be exported here

The observations for abstraction: TF

Abstraction	Register+start	Flight+start	ExpHotel+start	NoPickUp+st...	BookMore+st...	MedHotel+st...	Cruise+start	LowHotel+start	Confirm+start	Noninstalme...	NoPresent+s...	Finish+start	PickUp+sta
[Finish+start] -[Finish+start_0]	1000	2519	1670	2776	4074	1713	2555	1691	1000	501	661	1000	2298
[Register+start] -[Register+start_0]	1000	0	0	0	0	0	0	0	0	0	0	0	0
[Instalment+start Noninstalment+start] -[Instalment+start_0,Noninsta...	1000	2519	1670	2776	4074	1713	2555	1691	1000	501	206	0	2298
[NoPresent+start Present+start] -[NoPresent+start_0,Present+start_0]	1000	2519	1670	2776	4074	1713	2555	1691	1000	338	661	0	2298
[NoPickUp+start PickUp+start] -[NoPickUp+start_0,PickUp+start_0]	1000	497	329	552	0	339	503	332	0	0	0	0	448
[ExpHotel+start LowHotel+start MedHotel+start] -[ExpHotel+start_0,L...	1000	497	329	0	0	339	503	332	0	0	0	0	0
[BookMore+start Confirm+start] -[BookMore+start_0,Confirm+start_0]	1000	497	329	552	1000	339	503	332	0	0	0	0	448
[Cruise+start Flight+start] -[Cruise+start_0,Flight+start_0]	1000	497	0	0	0	0	503	0	0	0	0	0	0
[Register+start] -[Register+start_1]	0	2519	1670	2776	4074	1713	2555	1691	1000	501	661	1000	2298
[Cruise+start Flight+start] -[Cruise+start_0,Flight+start_1]	0	233	164	286	497	174	264	159	0	0	0	0	211
[ExpHotel+start LowHotel+start MedHotel+start] -[ExpHotel+start_1,L...	0	151	112	152	329	108	178	109	0	0	0	0	177
[NoPickUp+start PickUp+start] -[NoPickUp+start_1,PickUp+start_0]	0	268	182	302	552	181	284	189	0	0	0	0	250
[BookMore+start Confirm+start] -[BookMore+start_1,Confirm+start_0]	0	475	329	548	862	331	525	340	138	0	0	0	452
[Cruise+start Flight+start] -[Cruise+start_0,Flight+start_2]	0	102	74	123	199	75	97	84	34	23	34	34	110
[ExpHotel+start LowHotel+start MedHotel+start] -[ExpHotel+start_1,L...	0	94	71	109	188	71	94	46	34	19	34	34	113
[NoPickUp+start PickUp+start] -[NoPickUp+start_2,PickUp+start_0]	0	143	83	151	260	95	117	82	42	28	42	42	109
[BookMore+start Confirm+start] -[BookMore+start_2,Confirm+start_0]	0	432	285	483	714	309	430	268	148	0	0	0	379
[Cruise+start Flight+start] -[Cruise+start_1,Flight+start_2]	0	140	106	172	261	111	121	95	51	24	50	51	140
[ExpHotel+start LowHotel+start MedHotel+start] -[ExpHotel+start_1,L...	0	87	49	98	152	51	65	52	34	22	34	34	88
[NoPickUp+start PickUp+start] -[NoPickUp+start_3,PickUp+start_0]	0	59	51	71	133	31	74	51	18	7	18	18	62
[BookMore+start Confirm+start] -[BookMore+start_2,Confirm+start_1]	0	0	0	0	0	0	0	0	0	70	147	148	0
[Instalment+start Noninstalment+start] -[Instalment+start_0,Noninsta...	0	0	0	0	0	0	0	0	0	0	231	501	0
[NoPresent+start Present+start] -[NoPresent+start_1,Present+start_0]	0	0	0	0	0	0	0	0	0	104	0	661	0
[Finish+start] -[Finish+start_1]	0	0	0	0	0	0	0	0	0	0	0	0	0
[NoPickUp+start PickUp+start] -[NoPickUp+start_0,PickUp+start_1]	0	207	147	246	448	150	241	151	0	0	0	0	202

```
23:05:20 [D] [OpenXES]: NikeFS2: Allocating new swap file. (#1: 67108864 bytes)
23:05:21 [D] [OpenXES]: NikeFS2: Populating shadow map 1 (of 2 max.)
```

Figure D.3: View the mined log observation matrices for selected abstraction(s).

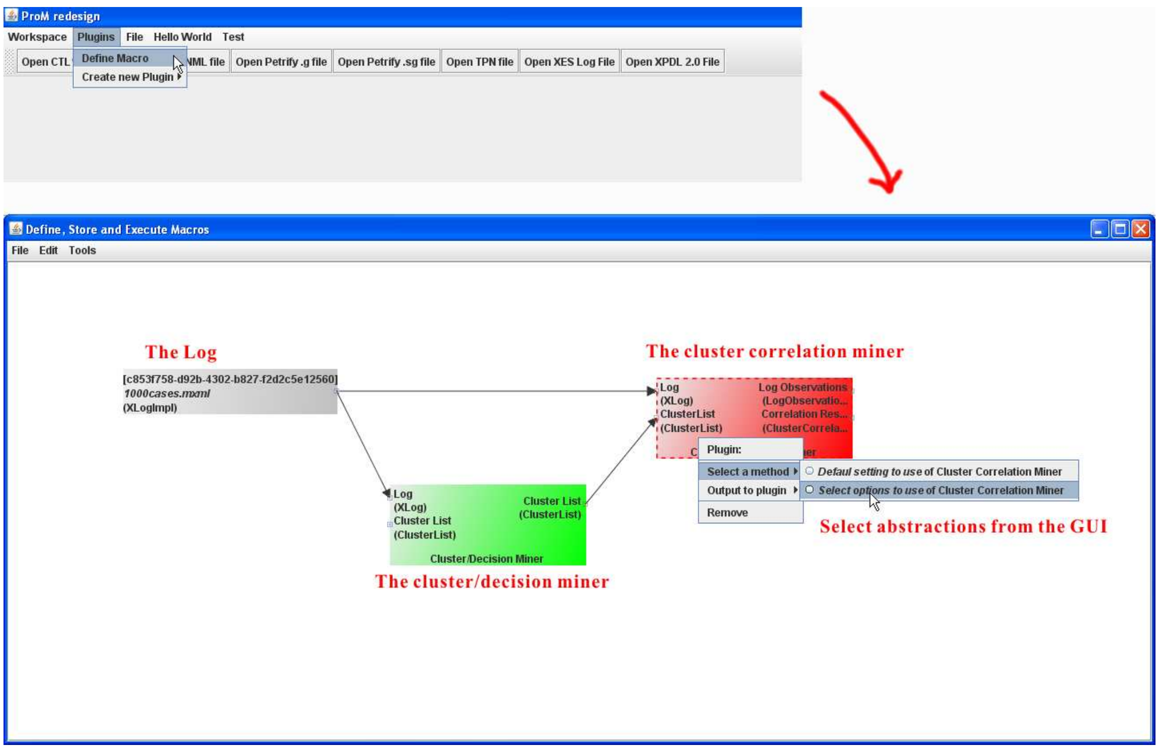


Figure D.4: Define a macro.

Bibliography

[1] *http://wiki.daimi.au.dk/cpntools-help/_home.wiki.*

[2] *org.apache.commons.math.mathexception.*

[3] *Standard ml of new jersey. www.smlnj.org.*

[4] A.Kolmogorov, *Moscow Univ. Math. Bull.*, 1 (1937).

[5] A. Bobbio, *System Modelling with Petri Nets*, (1990), 103–143.

[6] Christopher A. Chung, *Simulation Modeling Handbook: A Practical Approach*, CRC Press, Inc., 2003.

[7] Jorg Desel and Javier Esparza, *Free Choice Petri Nets*, Cambridge University Press, 1995.

[8] Frederick J Gravetter and Larry B. Wallnau, *Statistics for the Behavioral Sciences*, Wadsworth Publishing, 6 edition, 2003.

[9] Reiko Heckel, *Open Petri Nets as Semantic Model for Workflow Integration*, Petri Net Technology for Communication-Based Systems **Volume 2472** (2003), 281–294.

[10] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, Springer-Verlag.

[11] J.Neveu, *Mathematical foundations of the calculus of probability*, Holden-day (1965).

[12] W.M.P. van der Aalst M. Dumas and A.T. Hofstede, *Process-Aware Information Systems: Bridging People and Software Through Process Technology*, John Wiley and Sons, 2005.

[13] Douglas C. Montgomery and George C. Runger, *Applied Statistics and Probability for Engineers*, John Wiley and Sons, Inc.; 3rd edition, 2002.

[14] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall PTR, 1981.

[15] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst, *Discovering Simulation Models*, Inf. Syst. **34** (2009), no. 3, 305–327.

[16] H. Schonenberg, N. Sidorova, W. M. P. van der Aalst, and K. van Hee, *History-Dependent Stochastic Petri Nets*, in Post-proc. of the 7th Int. Conf. on Perspectives of System Informatics, PSI'2006 **accepted for publication** (2009), 305–317.

[17] W. M. P. van der Aalst, *Three Good Reasons for Using a Petri-net-based Workflow Management System*, (1996), 179–201.

[18] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters, *Workflow mining: A Survey of Issues and Approaches*, Data Knowl. Eng. **47** (2003), no. 2, 237–267.

[19] W.M.P. van der Aalst and Kees van Hee, *Workflow Management: Models, Methods, and Systems*, MIT Press, 2004.

[20] W.M.P. van der Aalst and M. Voorhoeve, *Business Process Simulation: lecture notes*, Technische Universiteit Eindhoven, 2008.

[21] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst, *The prom framework: A new era in process mining tool support*, Applications and Theory of Petri Nets 2005 (2005), 444–454.

[22] K.M. van Hee, A. Serebrenik, N. Sidorova, and W.M.P. van der Aalst, *History-Dependent Petri Nets*, **4546** (2007), 164–183.