# Business Process Configuration in The Cloud: How to Support and Analyze Multi-Tenant Processes?

W.M.P. van der Aalst
*Department of Mathematics and Computer Science, Eindhoven University of Technology*
*Eindhoven, The Netherlands*
*WWW: vdaalst.com*

***Abstract*—Lion's share of cloud research has been focusing on performance related problems. However, cloud computing will also change the way in which business processes are managed and supported, e.g., more and more organizations will be sharing common processes. In the classical setting, where product software is used, different organizations can make ad-hoc customizations to let the system fit their needs. This is undesirable, especially when multiple organizations share a cloud infrastructure. *Configurable process models* enable the sharing of common processes among different organizations in a controlled manner. This paper discusses challenges and opportunities related to business process configuration. *Causal nets* (C-nets) are proposed as a new formalism to deal with these challenges, e.g., merging variants into a configurable model is supported by a simple union operator. C-nets also provide a good representational bias for *process mining*, i.e., process discovery and conformance checking based on event logs. In the context of cloud computing, we focus on the application of C-nets to *cross-organizational* process mining.**

***Keywords*-configurable process models; cross-organizational process mining; cloud computing; causal nets**

## I. Introduction

The interest for *Cloud computing* is rapidly growing both in industry and research communities. Cloud computing focuses on the sharing of IT resources to achieve significant cost reductions. In this paper, we do not focus on the infrastructure level, but at the level of business processes shared among different parties. We aim to support families of processes for different organizations using the likes of Amazon Web Services (AWS) and Google App Engine (GAE).

*Multi-tenant processes* are organization-specific variants of the same process running in a cloud infrastructure. Consider for example Salesforce and Easychair. The sales processes of many organizations are managed and supported by Salesforce. On the one hand, these organizations share an infrastructure (processes, databases, etc.). On the other hand, they are not forced to follow a strict process model as the system can be configured to support variants of the same process. Easychair supports the review processes of many conferences. On the one hand, conferences share common functionality and processes. On the other hand, many variations are possible.

Despite examples like Salesforce and Easychair, *a systematic approach to support and analyze multi-tenant processes is missing*. Business Process Management (BPM) is gaining momentum, but for the actual implementation of most multi-tenant processes conventional methods are still being used. Processes and their configuration parameters are hard-coded and systems are data-centric rather than process-centric. This is undesirable and there is clear need for *process configuration* support [1], [2]. A configurable process model represents a family of process models. Through configuration, behavior is *restricted* to fit the needs of a particular organization.

Having a process-aware cloud infrastructure enables new forms of analysis. In this paper, we focus on *cross-organizational process mining* [3]. Process mining fuses data mining and process analysis techniques and is driven by the omnipresence of event data in modern information systems [4]. Cloud-based systems will record event logs in a systematic manner and at a scale not seen before. Moreover, event logs of different organizations can be analyzed and compared systematically.

We propose to use *Causal nets* (C-nets) as a formalism to support and analyze multi-tenant processes. C-nets are related to the representations used by several process discovery techniques (e.g., heuristic mining, fuzzy mining, and genetic mining) [4]. However, unlike mainstream notations like Petri nets, BPMN, BPEL, EPCs, and UML activity diagrams, C-nets provide declarative semantics more suitable for process mining and process configuration. For example, C-nets provide a much better representational bias than conventional techniques which allow for models that are obviously incorrect, i.e., by using C-nets the search space is reduced to the more interesting models. It is also very easy to merge different C-nets into an overarching C-net that allows for all behavior possible in the individual nets. This in stark contrast with more traditional notations for which it is far from trivial to fold a set of variants into a configurable model [5]. The verification of configurable models and their configurations is also difficult as such models implicitly describe families of models [6], [7].

The remainder is organized as follows. Section II introduces the topic of process configuration. Then we introduce C-nets as a new formalism to deal with the challenges imposed by process configuration and mining (Section III). Subsequently, we use C-nets to introduce the notions of process configuration (Section IV) and process mining (Section V). Section VI discusses challenges and opportunities related to multi-tenant processes. Section VII concludes this paper.

## II. Process Configuration

To motivate the need for configurable processes, we first sketch some example domains where many variants of the same process co-exist.

There are about 430 municipalities in The Netherlands. In principle, they all execute variants of the same set of processes [3], [5]. For example, they all support processes related to building permits, such as the process handling applications for permits and the process for handling objections against such permits.

Suncorp is the largest Australian insurance group. The Suncorp Group offers various types of insurance using brands such as Suncorp, AAMI, APIA, GIO, Just Car, Bingle, Vero, etc. There are insurance processes related to different types of risks (home, motor, commercial, liability, etc.) and these processes exist for the different Suncorp brands. Hence, there are up to 30 different variants of the process of handling an insurance claim at Suncorp.

Hertz is the largest car rental company in the world with more than 8,000 locations in 146 countries. All offices of Hertz need to support the same set of processes, e.g., how to process a reservation. However, there are subtle differences among the processes at different locations due to regional or national variations. For example, the law in one country or the culture in a particular region forces Hertz to customize the standard process for different locations.

Organizations such as Suncorp and Hertz need to support many variants of the same process (intra-organizational variation). Different municipalities in a country need to offer the same set of services to their citizens, and, hence, need to manage similar collections of processes. However, due to demographics and political choices, municipalities are handling things differently. Sometimes these differences are unintentional; however, often these differences can be easily justified by the desired "Couleur Locale" (inter-organizational variation). Clearly, it is undesirable to support these intra-organizational and inter-organizational variations by developing organization-specific systems or by adapting standard systems in an ad-hoc manner. Cloud technology can help to reduce costs by sharing a common infrastructure, but currently process configuration is not supported by such infrastructures.

As cloud infrastructures are gaining importance, it becomes more important to better support variability using configurable process models. Multi-tenant processes, i.e., the sharing of similar processes in the cloud, require a more systematic treatment of process configuration. Therefore, we elaborate on the nature of process configuration.

Michelangelo Buonarroti (1475–1564), the famous Italian sculptor, painter, architect and engineer, made the following well-known statements which illustrate the idea of configuration:

- "Every block of stone has a statue inside it and it is the task of the sculptor to discover it."
- "I saw the angel in the marble and carved until I set him free."
- "Carving is easy, you just go down to the skin and stop."

We take the viewpoint that *configuration is like sculpting*, i.e., carving out the desired behavior. This means that configuration implies the removal of behavior, i.e., blocking particular execution paths in the process.

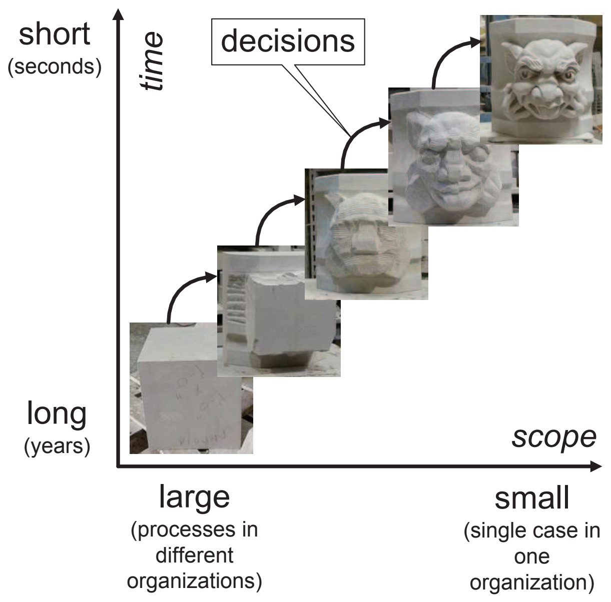
short
(seconds)
time
decisions
long
(years)
scope
large
(processes in different organizations)
small
(single case in one organization)

Figure 1. Configuration is like carving stone to create a sculpture. Making choices removes potential behavior, just like a sculptor removes stone. Decisions need to be made at different levels until at run-time all decisions have been made.

When developing product software that will be used in many organizations, one needs to make choices that will impact all of these organizations, the processes in these organizations, and the instances of these processes. Therefore, the scope of such design decisions is large and the timeframe associated with such decisions is long. Organizations using SAP R/3 benefit/suffer from choices made in the early 1990s when the ERP system was developed. When an organization chooses to install a system like SAP R/3, it needs to be configured to meet the specific needs of the organization. This implies that again various choices are made. The scope of such decisions is considerable, but smaller than the scope of initial design decisions made by the software vendor of a successful product. When configuring the system, one is operating within the bounds imposed by the product software. Subsequently, the installed system is used to support processes. This triggers another set of choices. Once the process is up and running, instances (often referred to as "cases") are handled by the system. However, there may still be choices left (cf. XOR-splits in a process model). These choices are resolved at run-time. When the instance has been handled, no choices remain. Sometimes, we refer to this as "audit time" as history cannot be changed and it is undesirable to try and change any records describing the completed execution of a process instance. Fig. 1 illustrates this process of decision making. At different levels, choices need to be made. Some choices have a small scope and a short timeframe whereas other choices have a large scope and a long timeframe. As shown in Fig. 1 there is a continuum of decision making. For example, one can have a process that is reconfigured when things get very busy. Imagine for example how Suncorp changed its processes when it got overloaded with thousands of claims due to the flooding of Queensland in January 2011. Such reconfiguration decisions impact many cases. One can also have a process that is different in weekends or during holidays. A process may depend on the region (e.g., location of a Hertz office), on the weather (e.g., when it rains a location is closed), on the type of customer (e.g., gold customers do not need to register), etc. All of these variations correspond to decisions that were made at some point in time. Decisions made at one level, remove options at a lower level.

In this paper, we assume that *configuration implies the removal of possibilities*. The desired behavior needs to be "carved out". Clearly, other viewpoints are possible. For example, some authors also consider refinement and model extension as configuration primitives [8]–[10]. Nevertheless, most approaches view configuration as the restriction of behavior before runtime. Moreover, when dealing with multi-tenant processes in the cloud, it seems unrealistic to assume that organizations will be able to remodel parts of standard processes.

Often two basic operators are used to remove behavior: "hiding" and "blocking" [11]. All other mechanisms for removing behavior can be expressed in terms of these basic operations (see [1], [2] for examples). Moreover, these correspond to the inverse of the basic operators describing inheritance of dynamic behavior [12]. While configuration corresponds to removing behavior, inheritance corresponds to adding behavior while preserving certain properties.

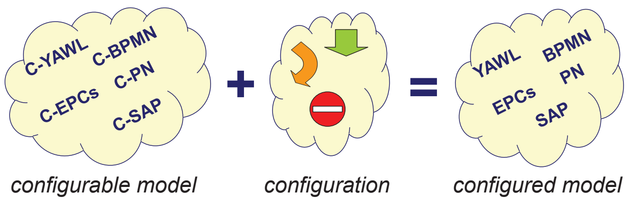

Figure 2. Given a configurable model and a configuration, a configured model is derived. The configured model has less behavior because potential behavior is removed during configuration.

Notions such as blocking and hiding are language independent. Fig. 2 illustrates that the basic mechanisms apply to different languages. Different configurable languages have been defined in the literature, e.g., C-YAWL, C-EPC, and C-SAP [1], [2]. A so-called *configuration* can be applied to a *configurable model* created using such a language (thus restricting behavior). After applying the configuration, one obtains the *configured model*. The latter is a conventional executable model, e.g., a model that can be enacted using a BPM system.

## III. Causal Nets

As illustrated by Fig. 2, there are various configurable languages where configuration corresponds to the removal of behavior. In this paper, we use *Causal nets* (C-nets) [4], [13], [14] as a basic language to reason about process configuration.

Fig. 3 shows a C-net modeling the booking of a trip. After activity $a$ (*start booking*) there are three possible activities: $b$ (*book flight*), $c$ (*book car*), and $d$ (*book hotel*). The process ends with activity $e$ (*complete booking*). Each activity has sets of potential *input* and *output bindings* (indicated by the black dots). Every connected set of dots on the output arcs of an activity is an output binding. For example, $a$ has four output bindings modeling that $a$ may be followed by (1) just $b$, (2) just $c$, (3) $b$ and $d$, or (4) $b$, $c$, and $d$. Hence, it is not possible to book just a hotel or a hotel and a car. Activity $c$ has two input bindings modeling that it is preceded by (1) just $a$ or (2) $a$ and $b$. This construct is used to model that when both a flight and a car are booked, the flight is booked first. Output bindings create *obligations* whereas input bindings remove obligations. For example, the occurrence of $a$ with output binding $\{b, d\}$ creates two obligations: both $b$ and $d$ need to be executed while referring to the obligations created by $a$.

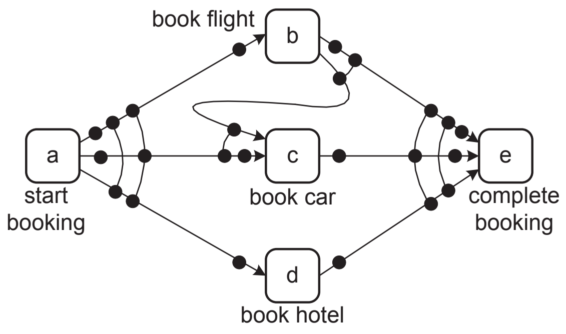

Figure 3. Causal net $C_{travel}$.

In a C-net there is one start activity ($a$ in Fig. 3) and one end activity ($e$ in Fig. 3). A *valid binding sequence* models an execution path starting with $a$ and ending with $e$ while removing all obligations created during execution. The behavior of a C-net is *restricted to valid binding sequences*. Hence, unlike conventional modeling languages, the semantics are non-local. The syntax of a C-net can be formalized as follows.

*Definition 1 (Causal net [4]):* A *Causal net* (C-net) is a tuple $C = (A, a_i, a_o, D, I, O)$ where:

- $A$ is a finite set of *activities*;
- $a_i \in A$ is the *start activity*;
- $a_o \in A$ is the *end activity*;
- $D \subseteq A \times A$ is the *dependency relation*,
- $AS = \{X \subseteq \mathcal{P}(A) \mid X = \{\emptyset\} \ \vee \ \emptyset \notin X\}$;[1]
- $I \in A \rightarrow AS$ defines the set of possible *input bindings* per activity; and

[1]$\mathcal{P}(A) = \{A' \mid A' \subseteq A\}$ is the powerset of $A$. Hence, elements of $AS$ are *sets of sets* of activities.

- $O \in A \to AS$ defines the set of possible *output bindings* per activity,

such that

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup_{as \in I(a_2)} as\}$;
- $D = \{(a_1, a_2) \in A \times A \mid a_2 \in \bigcup_{as \in O(a_1)} as\}$;
- $\{a_i\} = \{a \in A \mid I(a) = \{\emptyset\}\}$;
- $\{a_o\} = \{a \in A \mid O(a) = \{\emptyset\}\}$; and
- all activities in the graph $(A, D)$ are on a path from $a_i$ to $a_o$.

There is one unique start activity $a_i$ and one unique end activity $a_o$. Each activity $a$ has a set of possible input bindings $I(a)$ and a set of possible output bindings $O(a)$. For example, in Fig. 3, $I(c) = \{\{a\}, \{a, b\}\}$ and $O(c) = \{\{e\}\}$. Start activity $a_i$ has one input binding and end activity $a_o$ has one output binding: $I(a_i) = O(a_o) = \{\emptyset\}$. The notion of input and output bindings allows for the typical modeling constructs found in languages such as EPCs, BPMN, Petri nets, UML activity diagram, etc. Fig. 4 shows some of the basic process patterns.

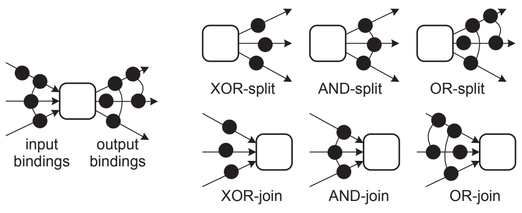

Figure 4. Input and output bindings.

An *activity binding* is a tuple $(a, as^I, as^O)$ denoting the occurrence of activity $a$ with input binding $as^I$ and output binding $as^O$. For example, $(c, \{a, b\}, \{e\})$ denotes the occurrence of activity $c$ in Fig. 3 while being preceded by $a$ and $b$, and succeeded by $e$.

*Definition 2 (Binding):* Let $C = (A, a_i, a_o, D, I, O)$ be a C-net. $B = \{(a, as^I, as^O) \in A \times \mathcal{P}(A) \times \mathcal{P}(A) \mid as^I \in I(a) \ \wedge \ as^O \in O(a)\}$ is the set of *activity bindings*. A *binding sequence* $\sigma$ is a sequence of activity bindings, i.e., $\sigma \in B^*$.

Note that sequences are denoted using angle brackets, e.g., $\langle \rangle$ denotes the empty sequence. $B^*$ is the set of all sequences over $B$ (including $\langle \rangle$). Function $\alpha \in B^* \to A^*$ projects binding sequences onto *activity sequences*, i.e., the input and output bindings are abstracted from and only the activity names are retained.

A possible binding sequence for the C-net shown in Fig. 3 is $\sigma = \langle (a, \emptyset, \{b, c, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{c, e\}), (c, \{a, b\}, \{e\}), (e, \{b, c, d\}, \emptyset) \rangle$, i.e., the scenario in which a hotel, a flight, and a car are booked. $\alpha(\sigma) = \langle a, d, b, c, e \rangle$ is the corresponding activity sequence. Note that in Fig. 3 a hotel can only be booked if a flight is booked. Moreover, when both a car and a flight are booked, then first the flight needs to be booked.

A binding sequence is *valid* if a predecessor activity and successor activity always "agree" on their bindings. For a predecessor activity $x$ and successor activity $y$ we need to see the following "pattern": $\langle \ldots, (x, \{\ldots\}, \{y, \ldots\}), \ldots, (y, \{x, \ldots\}, \{\ldots\}), \ldots \rangle$, i.e., an occurrence of activity $x$ with $y$ in its output binding needs to be followed by an occurrence of activity $y$, and an occurrence of activity $y$ with $x$ in its input binding needs to be preceded by an occurrence of activity $x$. To formalize the notion of a valid binding sequence, we first define the notion of *state*. States are represented by multi-sets of *obligations*, e.g., state $[(a, b)^2, (a, c)]$ denotes the state where there are two pending activations of $b$ by $a$ and there is one pending activation of $c$ by $a$. This means that $b$ needs to happen twice while having $a$ in its input binding and $c$ needs to happen once while having $a$ in its input binding.

*Definition 3 (State):* Let $C = (A, a_i, a_o, D, I, O)$ be a C-net. $S = \mathbb{B}(A \times A)$ is the *state space* of $C$.[2] $s \in S$ is a *state*, i.e., a multi-set of pending *obligations*. Function $\psi \in B^* \to S$ is defined inductively: $\psi(\langle \rangle) = [\,]$ and $\psi(\sigma \oplus (a, as^I, as^O)) = (\psi(\sigma) \setminus (as^I \times \{a\})) \uplus (\{a\} \times as^O)$ for any binding sequence $\sigma \oplus (a, as^I, as^O) \in B^*$.[3] $\psi(\sigma)$ is the state after executing binding sequence $\sigma$.

A *valid binding sequence* is a binding sequence that (1) starts with start activity $a_i$, (2) ends with end activity $a_o$, (3) only removes obligations that are pending, and (4) ends without any pending obligations. Consider, for example, the valid binding sequence $\sigma = \langle (a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{e\}), (e, \{b, d\}, \emptyset) \rangle$ for C-net $C_{travel}$ in Fig. 3:
$\psi(\langle \rangle) = [\,]$,
$\psi(\langle (a, \emptyset, \{b, d\}) \rangle) = [(a, b), (a, d)]$,
$\psi(\langle (a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}) \rangle) = [(a, b), (d, e)]$,
$\psi(\langle (a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{e\}) \rangle) = [(b, e), (d, e)]$,
$\psi(\langle (a, \emptyset, \{b, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{e\}), (e, \{b, d\}, \emptyset) \rangle) = [\,]$.
Sequence $\sigma$ indeed starts with start activity $a$, ends with end activity $e$, only removes obligations that are pending (i.e., for every input binding there was an earlier output binding), and ends without any pending obligations: $\psi(\sigma) = [\,]$.

*Definition 4 (Valid):* Let $C = (A, a_i, a_o, D, I, O)$ be a C-net and $\sigma = \langle (a_1, as^I_1, as^O_1), (a_2, as^I_2, as^O_2), \ldots, (a_n, as^I_n, as^O_n) \rangle \in B^*$ be a binding sequence. $\sigma$ is a *valid binding sequence* of $C$ if and only if:

- $a_1 = a_i$, $a_n = a_o$, and $a_k \in A \setminus \{a_i, a_o\}$ for $1 < k < n$;
- $\psi(\sigma) = [\,]$; and
- for any non-empty prefix $\sigma' = \langle (a_1, as^I_1, as^O_1), \ldots, (a_k, as^I_k, as^O_k) \rangle$ $(1 \leq k \leq n)$: $(as^I_k \times \{a_k\}) \leq \psi(\sigma'')$ with $\sigma'' = \langle (a_1, as^I_1, as^O_1), \ldots, (a_{k-1}, as^I_{k-1}, as^O_{k-1}) \rangle$

$V(C)$ is the *set of all valid binding sequences* of $C$.

The first requirement states that valid binding sequences start with $a_i$ and end with $a_o$ ($a_i$ and $a_o$ cannot appear

[2] $\mathbb{B}(A)$ is a multiset over $A$. $X = [a^2, b^3, c]$ is a multiset with six elements: two times $a$, three times $b$, and one $c$, $X \uplus Y$ is the union of two multi-sets. $X \setminus Y$ removes $Y$ from $X$ (difference of two multi-sets). Ordinary sets will be used as multi-sets throughout this paper.

[3] $\oplus$ is used to concatenate an element to the end of a sequence, e.g., $\langle a, b, c \rangle \oplus d = \langle a, b, c, d \rangle$.

in the middle of valid sequence). The second requirement states that at the end there should not be any pending obligations. (One can think of this as the constraint that no tokens left in the net.) The last requirement considers all non-empty prefixes of $\sigma$. The last activity binding of the prefix (i.e., $(a_k, as^I_k, as^O_k)$) should only remove pending obligations, i.e., $(as^I_k \times \{a_k\}) \leq \psi(\sigma'')$ where $as^I_k \times \{a_k\}$ are the obligations to be removed and $\psi(\sigma'')$ are the pending obligations just before the occurrence of the $k$-th binding. (One can think of this as the constraint that one cannot consume tokens that have not been produced.)

The C-net in Fig. 3 has seven valid binding sequences: only $b$ is executed ($\langle(a, \emptyset, \{b\}), (b, \{a\}, \{e\}), (e, \{b\}, \emptyset)\rangle$), only $c$ is executed (besides $a$ and $e$), $b$ and $d$ are executed (two possibilities), and $b$, $c$ and $d$ are executed (3 possibilities because $b$ needs to occur before $c$).

For the semantics of a C-net we only consider valid binding sequences, i.e., *invalid sequences are not part of the behavior* described by the C-net. This means that C-nets do not use plain "token-game semantics" as employed in conventional languages like BPMN, Petri nets, EPCs, and YAWL. The semantics of C-nets are more declarative as they are defined over complete sequences rather than a local firing rule. Note that the semantics abstract from the moment of choice; pending obligations are not exposed to the environment and are not fixed during execution (i.e., all *valid* interpretations remain open).

## IV. Configuration = Removing Behavior

After introducing C-nets as a process modeling language, we now focus on process configuration. As discussed in Section II, we restrict behavior to configure a process. Whereas a *configurable model* allows for the behavior exhibited in different variants of a process, a *configured model* is more specific. To formalize the notion of configuration, we introduce some basic operations on C-nets.

*Definition 5 (Operations on C-nets):* Let $C_1 = (A_1, a_i, a_o, D_1, I_1, O_1)$ and $C_2 = (A_2, a_i, a_o, D_2, I_2, O_2)$ be two C-nets having identical start and end activities.

- $C_1 \sqsubseteq C_2$ if and only if $A_1 \subseteq A_2$ and for all $a \in A_1$: $I_1(a) \subseteq I_2(a)$ and $O_1(a) \subseteq O_2(a)$.
- $C_1 \uplus C_2 = (A, a_i, a_o, D, I, O)$ with $A = A_1 \cup A_2$, $D = D_1 \cup D_2$, $I = I_1 \sqcup I_2$, and $O = O_1 \sqcup O_2$.[4]

Consider two C-nets such that $C_1 \sqsubseteq C_2$. $C_1$ can be viewed as the configured model and $C_2$ as the configurable model. The intuition is that $C_1$ allows for less behavior than $C_2$, i.e., by removing potential input and output bindings the behavior of the process is restricted. It is possible to use more sophisticated notions. For example, one could define a configurable model as a set of C-nets, e.g., $\mathcal{C} = \{C \sqsubseteq C_b \mid B(C)\}$ where $B$ is a boolean expression over all C-nets that are contained in some base model $C_b$. $B$ can be used to encode domain constraints or data dependencies, e.g., activity $x$ may require input data produced by activity $y$, so $y$ cannot be hidden or blocked when $x$ is enabled [1], [6], [7]. For simplicity, we assume that $B$ evaluates to true in this paper, i.e., the set $\mathcal{C}$ is uniquely identified by some base model $C_b$. In the remainder, we will refer to $C_b$ as the configurable model and to $C \sqsubseteq C_b$ as one of its possible configurations.

*Theorem 1:* Let $C_1$ and $C_2$ be two C-nets having identical start and end activities. If $C_1 \sqsubseteq C_2$, then $V(C_1) \subseteq V(C_2)$.

*Proof:* Consider valid binding sequence $\sigma = \langle(a_1, as^I_1, as^O_1), (a_2, as^I_2, as^O_2), \ldots, (a_n, as^I_n, as^O_n)\rangle \in V(C_1)$. First we show that $\sigma$ is indeed a binding sequence of $C_2$, i.e., for any $k$: $(a_k, as^I_k, as^O_k)$ should be a binding. This is the case because $a_k \in A_1 \subseteq A_2$, $as^I_k \in I_1(a_k) \subseteq I_2(a_k)$ and $as^O_k \in O_1(a_k) \subseteq O_2(a_k)$. Clearly, $a_1 = a_i$, $a_n = a_o$, and $a_k \in A_1 \setminus \{a_i, a_o\} \subseteq A_2 \setminus \{a_i, a_o\}$ for $1 < k < n$, i.e., the first requirement in Definition 4 is satisfied. $\psi(\sigma) = [\,]$ in both models because this function only depends on $\sigma$ and not on the model. The same holds for the third requirement in Definition 4. Hence, $\sigma \in V(C_2)$. ■

Theorem 1 shows that the set of valid sequences of the configured process $C_1$ is included in the set of valid sequences of the configurable process $C_2$. Fig. 5 shows three C-nets whose behaviors are embedded in the initial C-net of Fig. 3. It is easy to see that $C_1 \sqsubseteq C_{travel}$, $C_2 \sqsubseteq C_{travel}$, and $C_3 \sqsubseteq C_{travel}$. This implies that all valid binding sequences of these three variants are included in the set of valid binding sequences of $C_{travel}$.

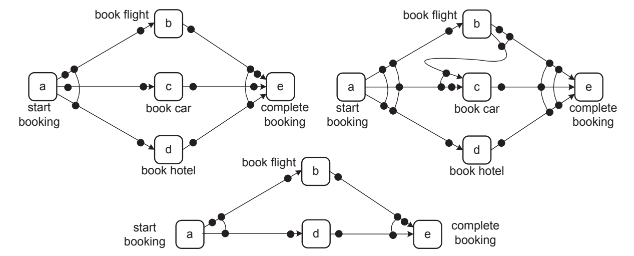

Figure 5. Three configured process models obtained by restricting the configurable process model in Fig. 3: $C_1$ (top left), $C_2$ (top right), and $C_3$ (bottom). Note that $C_{travel} = C_1 \uplus C_2 \uplus C_3$.

A configurable process model contains the behavior of different variants. This is operationalized by the following observation.

*Corollary 1:* Let $C_1$ and $C_2$ be two C-nets having identical start and end activities. $V(C_1) \cup V(C_2) \subseteq V(C_1 \uplus C_2)$.

The corollary follows directly from Theorem 1 because $C_1 \sqsubseteq C_1 \uplus C_2$ and $C_2 \sqsubseteq C_1 \uplus C_2$. Using classical process notations such as Petri nets, EPCs, BPMN, UML activity diagrams, it is far from trivial to merge models. The merged model may have all kinds of anomalies (deadlocks, livelocks, etc.) as discussed [5], [7]. Using the declarative semantics of C-nets, it is trivial to merge variants of a process into a model that allows for all behaviors present in the individual variants.

[4]$\sqcup$ takes the union of two set valued functions, i.e., $f = f_1 \sqcup f_2$ is a function such that $f(x) = f_1(x)$ if $x \in dom(f_1) \setminus dom(f_2)$, $f(x) = f_2(x)$ if $x \in dom(f_2) \setminus dom(f_1)$, and $f(x) = f_1(x) \cup f_2(x)$ if $x \in dom(f_1) \cap dom(f_2)$.

For C-nets it is easy to add and remove behavior and to merge process variants. As indicated in Section I, this is important in the context of multi-tenant processes. The execution can simply block input and output bindings for particular tenants.

## V. Mining = Discovering Behavior

The goal of process mining is to extract knowledge about a particular (operational) process from event logs. An *event* refers to the occurrence of an activity for a particular *process instance* (often referred to as *case*). The events related to a case are ordered and form a *trace*.

*Definition 6 (Event, Trace, Event log):* Let $A$ be a set of activities. $\sigma \in A^*$ is a *trace*, i.e., a sequence of events. $L \in \mathbb{B}(A^*)$ is an *event log*, i.e., a multi-set of traces.

An event log may contain additional information, e.g., an event may have an explicit timestamp and attributes such as amount, customer, resource, etc. As shown in [4], this information can be used for different types of analysis ranging from bottleneck analysis and prediction to social network analysis. For simplicity we abstract from this and consider just activity names. Moreover, we focus on C-nets as a target model. This allows us to restrict the notion of process mining to the discovery of a C-net from a multiset of traces.

*Definition 7 (Mining algorithm):* Let $L$ be an event log. A mining algorithm $\mu$ derives a process model from such an event log, i.e., $\mu(L) = (A, a_i, a_o, D, I, O)$ when C-nets are used as a target model.

Process discovery algorithms can discover models such as the one shown in Fig. 3 from event logs such as $L_{travel} = [\langle a,b,e\rangle^8, \langle a,c,e\rangle^{11}, \langle a,b,d,e\rangle^{30}, \langle a,d,b,e\rangle^{22}, \langle a,b,c,d,e\rangle^7, \langle a,b,d,c,e\rangle^{12}, \langle a,d,b,c,e\rangle^9]$, i.e., $\mu(L_{travel}) = C_{travel}$.

Various process discovery algorithms have been proposed in literature (see [4] for an overview). These algorithms use techniques varying from genetic algorithms to region-based approaches. There are different ways to define the quality of the discovered process model, e.g., *fitness* measures the fraction of the event log that can be "replayed" by the event log.

*Definition 8 (Fitness):* Let $L$ be an event log and $C$ a discovered process model. A fitness function $\pi$ measures the quality of the model with respect to the log. For example, a naive fitness function is $\pi(L, C) = \#[\sigma \in L \mid \exists_{\sigma' \in V(C)} \alpha(\sigma') = \sigma] / \#L$.

Various fitness notions have been proposed. The example function $\pi(L, C)$ given in Definition 7 simply measures the fraction of traces that correspond to valid binding sequences. A single deviating event in a long trace that has no other deviations results in a non-fitting case. Hence, the metric cannot distinguish between an almost fitting trace and a trace consisting of many deviating events. Therefore, most notions measure fitness at the level of events [4], [15]. Moreover, fitness is just one dimension. In general, there is a trade-off between the following four quality criteria: (a) *fitness*: the discovered model should allow for the behavior seen in the event log, (b) *precision*: the discovered model should not allow for behavior completely unrelated to what was seen in the event log, (c) *generalization*: the discovered model should generalize the example behavior seen in the event log, and (d) *simplicity*: the discovered model should be as simple as possible.

Traditionally, process mining techniques are applied to one event log originating from some process, i.e., the focus is on a single log. In the context of configurable/multi-tenant processes, we need to compare the event logs and models of different organizations. As clouds provide an infrastructure to systematically collect events, cross-organizational analysis comes into reach.

## VI. Challenges and Opportunities

After introducing C-nets as a representation for process configuration and process mining, we now discuss the challenges and opportunities associated to multi-tenant processes in the cloud. Fig. 6 is used to structure the discussion. Assume there are $n$ variants of the same process running in parallel. We refer to these processes as $P_1, P_2, \ldots P_n$. These processes generate event logs $L_1, L_2, \ldots L_n$. The corresponding process models are $C_1, C_2, \ldots C_n$. A process model $C_i$ may have been made by hand or has been discovered from event log $L_i$ (for $1 \leq i \leq n$). In the latter case: $C_i = \mu(L_i)$. To simplify the discussion, we assume that $C_i$ is a C-net as defined in Definition 1 and $L_i$ is an event log as defined in Definition 6. Note that in reality event logs will contain much more information and process models also describe resources, data flow, routing conditions, triggers, etc. [4]. Fig. 6 shows for each process $P_i$ an event log $L_i$ and process model $C_i$. The different process models can be merged into an overall configurable model $C_{all} = C_1 \uplus C_2 \uplus \ldots \uplus C_n$. The overall event log covering all variants can be obtained my merging the individual event logs: $L_{all} = L_1 \uplus L_2 \uplus \ldots \uplus L_n$. An alternative way to get the overall configurable model is via discovery based on the merged event log, i.e., $C_{all} = \mu(L_{all})$. Assuming that the processes have been running for a while, it is also possible to extract Key Performance Indicators (KPIs) and compare these among different variants. The small meter-like symbols in Fig. 6 refer to these KPIs.

### A. Capturing Variability into a Single Configurable Model

As indicated in Section IV, a configurable model describes a family of processes $\mathcal{C} = \{C \sqsubseteq C_b \mid B(C)\}$ where $B$ is a boolean expression over a C-nets that are contained in some base model $C_b$. To simplify discussion, we referred to $C_b$ as if it is the configurable model. In reality, it is more difficult to describe a configurable model. First of all, the model should also take other perspectives into account. Second, it is non-trivial to describe the possible configurations in an intuitive manner. Consider for example a set of rather sequential process variants that do not agree on the ordering of most activities. The resulting model will be spaghetti-like and it is still unclear how to address this problem properly [1], [6], [7].

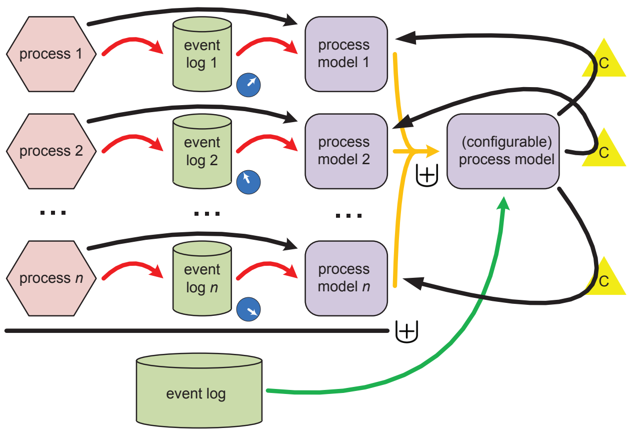

Figure 6. Overview of the different challenges and connections between configurable/configured models and event logs.

## B. Analyzing Configurable Process Models

Traditionally, analysis techniques focus on the analysis of a single process model. Configurable process models define a family of process models. Therefore, analysis becomes more challenging. For example, in [7] we show how to compute a so-called "configuration guideline" that allows for the analysis of potential problems at design time. The configuration guideline also enables new functionality such as "auto-completing" a partially configured model (taking data dependencies into account).

## C. Learning Configurable Models

As Fig. 6 shows there are basically two ways to obtain a configurable model. One approach is to merge the different models: $C_{all} = C_1 \uplus C_2 \uplus \ldots \uplus C_n$. Another approach is to merge the individual event logs into one log $L_{all} = L_1 \uplus L_2 \uplus \ldots \uplus L_n$ and then apply some process discovery algorithm $C_{all} = \mu(L_{all})$. For C-nets this is relatively simple provided that the various "ingredients" (event logs, models, etc.) are available and of high quality. In practice, these assumptions often do not hold, thus making the problem challenging. Moreover, it will rarely be the case that $\mu(L_{all})$ or $C_1 \uplus C_2 \uplus \ldots \uplus C_n$ will be taken as the definitive configurable model as the goal is to capture "best practices". Undesirable features of particular processes may be left out or processes are unified to "fight complexity".

## D. Conformance Checking

Given an event log $L$ and a process model $C$, we can check the conformance of both. For example, we can measure fitness $\pi(L, C)$, i.e., the fraction of the event log that is possible according to the model (can be defined in different ways [4]). In the phase where one is building a configurable model $C$, it is interesting to see how good $C$ fits the event log of each individual organization, i.e., measure $\pi(L_i, C)$ for $1 \leq i \leq n$. This will show which organizations are most affected by (changes to) the configurable model. Note that the goal is not to simply merge all existing behaviors; the configurable model should capture "best practices" while still allowing for the "Couleur Locale" required by the individual organizations.

Conformance checking techniques [4], [15] analyze behavioral differences between a log and model, i.e., *log-to-model* comparison. However, it is also possible to compare two models or two logs. *Model-to-model* comparison can be used to analyze the differences between the individual process models $C_1, C_2, \ldots C_n$ or differences between the configurable model $C$ and a specific process model $C_i$. Log-to-log comparison can be used to compare the different processes based on their event logs. For example, one can find that $L_i$ and $L_j$ are most similar or that the behavior of $L_i$ is contained in $L_j$.

## E. Cross-Tenant Process Mining

Having event logs from different organizations in a format that allows for comparison, enables cross-organizational process mining. The service provider that is supporting processes for different organizations (e.g. Salesforce or Easychair) is interested in knowing how organizations are using the common infrastructure. Comparative analysis may lead to additional services (e.g., guidance of end-user organizations) or to changes to the configurable model. In the CoSeLog project [3], [16], [17] we experienced that organizations are interested in learning from each other. In this project, 10 of the 430 Dutch municipalities are participating to investigate how cloud technology can be used to reduce costs and improve service. All Dutch municipalities need to offer the same services to their citizens, and need to manage similar collections of processes. However, due to demographics and political choices, municipalities are handling things differently. Therefore, process configuration is important. Moreover, municipalities are not in competition with one another. In fact, the ten municipalities involved in CoSeLoG are eager to learn "proven best practices" from one another. This can be operationalized using cross-organizational process mining [17].

Besides a pair-wise comparison of logs and models, we can also use supervised learning to explain differences. For example, we can use classification techniques such as decision tree learning. For this purpose we need to label the data at the level of cases or at the level of event logs. Classification is based on a selected *response variable* and a set of *predictor variables*. For example, the response variable could be the (average) flow time or costs of a case or log. The fitness of an event log or case with respect to some reference model can also be taken as a response variable. Predictor variables are other properties of cases, event logs, or process models. For example, the complexity of the process model and the number or resources involved. Based on such information one can construct a decision tree that aims to *explain the response variable in terms of predictor variables*. For example, classification based on logs of different municipalities may reveal that (a) larger municipalities tend to have fewer deviations, (b) allowing for more concurrency results in shorter flow times but more deviations, and (c) a pre-check of building permits results in shorter flow times and a higher acceptance rate.

## VII. Conclusion

Emerging cloud infrastructures provide new opportunities for sharing processes and allow for new types of analysis such as cross-organizational process mining. When organizations share processes in the cloud, we refer to these as multi-tenant processes. However, different organizations have different requirements. Therefore, the cloud infrastructure needs to support variants of the same process. In product software installed locally (e.g., SAP), it is common practice to customize the system in a rather ad-hoc manner. Clearly, this is unacceptable in a cloud setting. Therefore, we advocated the need for configurable process models. Once different variants of the same configurable model are running in the cloud, events are recorded in a systematic manner and it becomes relatively easy to compare these variants using cross-organizational mining.

In this paper, we advocated the use of Causal nets (C-nets) for process configuration and mining. This representation emerged from the process mining domain [4]. As shown, essential operators such as merging two models $C_1 \uplus C_2$ and checking containment $C_1 \sqsubseteq C_2$ can be defined without the traditional complications (e.g., dealing with deadlock and other anomalies). Although C-nets are close to existing languages having AND/XOR/OR-splits/joins, their semantics are defined in terms valid binding sequences. These semantics simplify both configuration and process mining.

## Acknowledgments

The author would like to thank Marcello La Rosa, Florian Gottschalk, Niels Lohmann, Michael Rosemann, Jan Vogelaar, Marlon Dumas, Eric Verbeek, Boudewijn van Dongen, Joos Buijs, and Hajo Reijers for their work on configurable models (C-YAWL, C-EPC, etc.) and their involvement in the CoSeLoG project.

## References

[1] M. Rosemann and W. van der Aalst, "A Configurable Reference Modelling Language," *Information Systems*, vol. 32, no. 1, pp. 1–23, 2007.

[2] F. Gottschalk, W. van der Aalst, M. Jansen-Vullers, and M. L. Rosa, "Configurable Workflow Models," *International Journal of Cooperative Information Systems*, vol. 17, no. 2, pp. 177–221, 2008.

[3] W. van der Aalst, "Configurable Services in the Cloud: Supporting Variability While Enabling Cross-Organizational Process Mining," in *CoopIS 2010*, Lecture Notes in Computer Science, vol. 6426. Springer-Verlag, Berlin, 2010, pp. 8–25.

[4] W. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.

[5] F. Gottschalk, T. Wagemakers, M. Jansen-Vullers, W. van der Aalst, and M. La Rosa, "Configurable Process Models: Experiences From a Municipality Case Study," in *CAiSE'09*, Lecture Notes in Computer Science, vol. 5565. Springer-Verlag, Berlin, 2009, pp. 486–500.

[6] W. van der Aalst, M. Dumas, F. Gottschalk, A. ter Hofstede, M. La Rosa, and J. Mendling, "Preserving Correctness During Business Process Model Configuration," *Formal Aspects of Computing*, vol. 22, no. 3, pp. 459–482, 2010.

[7] W. van der Aalst, N. Lohmann, M. La Rosa, and J. Xu, "Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis," in *Business Process Management (BPM 2010)*, ser. Lecture Notes in Computer Science, R. Hull, J. Mendling, and S. Tai, Eds., vol. 6336. Springer-Verlag, Berlin, 2010, pp. 95–111.

[8] J. Becker, P. Delfmann, and R. Knackstedt, "Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models," in *Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models*, Physica-Verlag, Springer, 2007, pp. 27–58.

[9] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing Variability in Business Process Models: The Provop Approach," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 22, no. 6-7, pp. 519–546, 2010.

[10] I. Reinhartz-Berger, P. Soffer, and A. Sturm, "Extending the Adaptability of Reference Models," *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, vol. 40, no. 5, pp. 1045–1056, 2011.

[11] F. Gottschalk, W. van der Aalst, and M. Jansen-Vullers, "Configurable Process Models: A Foundational Approach," in *Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models*, Physica-Verlag, Springer, 2007, pp. 59–78.

[12] W. van der Aalst and T. Basten, "Inheritance of Workflows: An Approach to Tackling Problems Related to Change," *Theoretical Computer Science*, vol. 270, no. 1-2, pp. 125–203, 2002.

[13] W. van der Aalst, "Do Petri Nets Provide the Right Representational Bias for Process Mining?" in *Workshop Applications of Region Theory 2011 (ART 2011)*, ser. CEUR Workshop Proceedings, J. Desel and A. Yakovlev, Eds., vol. 725. CEUR-WS.org, 2011, pp. 85–94.

[14] W. van der Aalst, A. Adriansyah, and B. van Dongen, "Causal Nets: A Modeling Language Tailored Towards Process Discovery," in *22nd International Conference on Concurrency Theory (CONCUR 2011)*, ser. Lecture Notes in Computer Science, J. Katoen and B. Koenig, Eds. Springer-Verlag, Berlin, 2011.

[15] A. Rozinat and W. van der Aalst, "Conformance Checking of Processes Based on Monitoring Real Behavior," *Information Systems*, vol. 33, no. 1, pp. 64–95, 2008.

[16] CoSeLoG, "Configurable Services for Local Governments (CoSeLoG) Project Home Page," www.win.tue.nl/coselog.

[17] J. Buijs, B. van Dongen, and W. van der Aalst, "Towards Cross-Organizational Process Mining in Collections of Process Models and their Executions," in *International Workshop on Process Model Collections (PMC 2011)*. Springer-Verlag, Berlin, 2011.