Systems Mining with HERAKLIT: The Next Step

Peter Fettke[1,2(✉)] and Wolfgang Reisig[3]

[1] German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
peter.fettke@dfki.de
[2] Saarland University, Saarbrücken, Germany
[3] Humboldt-Universität zu Berlin, Berlin, Germany
reisig@informatik.hu-berlin.de

Abstract. We suggest *systems mining* as the next step after process mining. Systems mining starts with a more careful investigation of runs, and constructs a detailed model of behavior, more subtle than classical process mining. The resulting model is enriched with information about data. From this model, a system model can be deduced in a systematic way.

Keywords: Systems composition · Data modeling · Behavior modeling · Composition calculus · Algebraic specification · Systems mining

1 Introduction

Classical process mining methods as established in theory and practice start out with *event logs*, generated by *processes* during their dynamic progression [1,7]. Process mining is designed first of all to discover processes by extracting knowledge from event logs. Each event in an event log is conceived as an activity that has been performed in the process at the point in time given in the event log, and is related to a particular case. Typically, the events of a case are totally or weakly ordered and can be seen as an execution or run of the process.

The left side of Fig. 1 depicts the standard formal approach for understanding an event log, a processes model, and a system, namely, behavior can be understood as three different sets of symbol sequences [3]. In this paper, we propose to follow a different route: there is no reason to assume that the events of a run are totally or weakly ordered. Of course, a clock outside the run may timestamp a run's events. This induces an order; however, this order is irrelevant for a proper understanding of a run. To the contrary, it spoils the *causal* order of events, which orders two events a and b by $a < b$ if and only if a is a prerequisite for b. Of course, $a < b$ implies each potential clock to timestamp a before b. But a timestamped before b only implies that b is not a prerequisite for a. Or, in one sentence: causality matters!

Additionally, systems to be mined are typically not monolithic, amorphous or unstructured, but can best be described and understood as the composition
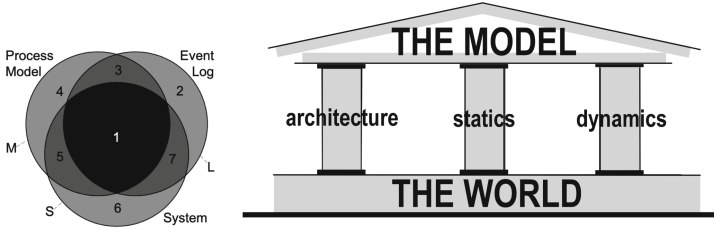
Fig. 1. Classical process mining (left, source: [3]) and systems mining.

of different sub-systems. Hence, an understanding of the different modules of a system is necessary while understanding the behavior of the system. Again, in one sentence: composition matters!

Last, but not least, a system processes data. Data processing is not only needed for the correct execution of processes, but also for the symbolic representation of important objects, e.g. invoices, customers, agreements, orders, products, and many more objects of interest. These objects need to be understood and represented adequately while mining a system. Again, in one sentence: objects matter!

In this contribution, by means of an example, we show how to mine not only process models, but entire system models. This includes the integrated modeling of architecture, statics, and dynamics of the world we live in (Fig. 1, right). To this end, we combine the well-known techniques of Petri nets and abstract data types with the recent *composition calculus*.

We motivate and exemplify a different notion of *runs*, by means of a case study from the area of retail sale. Supported by some static aspects of a system, such runs can be deduced from the system's event logs. Note, that our paper is purely conceptual. We do not provide an algorithm nor a software tool for systems mining. Instead, our main contribution is the elaboration of the new idea of systems mining based on the formal framework of HERAKLIT [5,6].

This paper starts with the presentation of the main idea of modules and runs while unfolding a running case study (Sect. 2). Sect. 3 describes systems nets and Sect. 4 presents the main idea for mining a system module. Related work is discussed in Sect. 5, Sect. 6 presents some conclusions.

2   Modules and Their Composition

Before presenting the (not too heavy) formal framework, we discuss a motivating example that later will be extended to a full case study.

2.1   Example: Occurrence Modules of a Retail Business

We start with a small log, recording observations from the field, namely seven events from a retail shop, as Fig. 2 shows. Each event has a unique name, a set of

Name of the event	involved agents	involved data	time stamp	Name of the event	involved agents	involved data	time stamp	Name of the event	involved agents	involved data	time stamp
shirt to take home	vendor V1 Alice	shirt	1	Alice pays	cashier Alice	50€ receipt	3	Bob pays	cashier Bob	170€ voucher	4
shoes to be ordered	vendor V2 Bob	shoes	2	hat not on offer	vendor V2 Claire	-	4	handing over	vendor V1 Alice	receipt packed shirt	5
V1 packs shirt	vendor V1	shirt	1								

Fig. 2. Event log with seven events.

involved agents, a set of data, and a timestamp. Static inspection of the system and the events of the log identifies *six agents*: Two vendors $V1$ and $V2$, a cashier, and three clients, Alice, Bob, and Claire. All events of the log, up to $V1$ *packs shirt*, include two agents. For example, the *shirt to take home* event includes the vendor $V1$ and the client Alice, jointly selecting a shirt for Alice. The *shoes to be ordered* event includes the vendor $V2$ and the client Bob, jointly agreeing on shoes, to be ordered from wholesale. The other events are intuitively obvious.

From the perspective of agents it is intuitively obvious that for a given event log, an agent is involved in a *sequence* of events, describing one of the potential *behaviors* of the agent. For example, the event log in Fig. 2 implies the vendor $V1$ be involved in three events: *shirt to take home*, *V1 packs shirt*, and *handing over*. This behavior can automatically be deduced from the log. An event updates the local state prior to its occurrence, and produces a local state as a result of its occurrence. Technically, we represent this as a Petri net, as in Fig. 3 (a). Each place (circle) denotes a local state; each transition (rectangle) denotes a step. An agent's behavior deduced from a log is very simple in structure; it can be thought of as a classical sequence of states and steps.

Figure 3 shows the behaviors of all six agents, deduced from the log in Fig. 2. Obviously, they are tightly interrelated, and this interrelation is now to be constructed explicitly. To do this, each behavior is embedded into a *module* in which each transition is either inside the module, or in an *interface* of the module. Each module has a *left* and a *right* interface. Graphically, a module is enclosed in a rectangle with the left and right interface elements on the left and right margin, respectively. The left and right interface of a module A is designated $^*A$ and A^*, respectively. This way, Fig. 4 (a) shows the module $V2$ of the vendor $V2$. The two transitions of the module are both located on the right interface, $V2^*$. The *Claire* module in Fig. 4 (b) places the transition of Claire's module on the left interface, $^*Claire$.

The two modules are now composed into a new module, $V2 \bullet Claire$, shown in Fig. 4 (c). To compose $V2$ and *Claire*, we merge the transition with label *hat not on offer* of $V2^*$ with the equally labeled transition of $^*Claire$. The resulting transition goes inside $V2 \bullet Claire$. The transition with label *shoes to be ordered* of $V2^*$ goes to $(V2 \bullet Claire)^*$.

This example shows the general principle of the composition of two modules A and B: Equally labeled elements of A^* and $^*B$ are merged and go into the interior of $A \bullet B$. The other elements from A^* and $^*B$ go to $(A \bullet B)^*$ and $^*(A \bullet B)$,
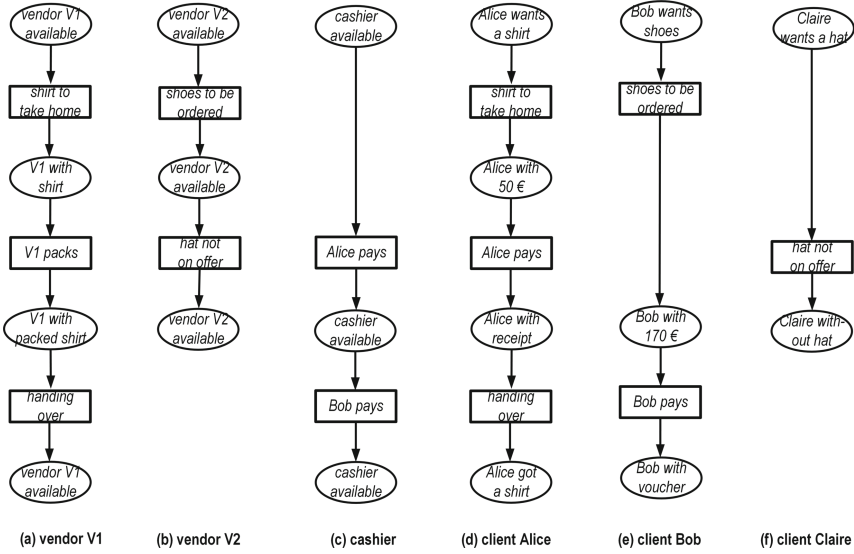
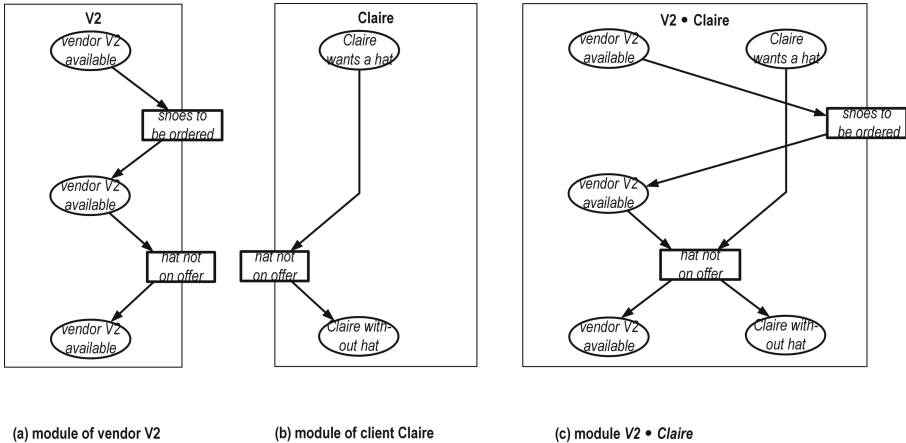**Fig. 3.** Agents and their behavior, elicited from the event log.

Fig. 4. Two modules and their composition.

respectively. This kind of composition motivates the distinction of right and left interfaces: The running example exhibits an intuitive dichotomy between shop modules (vendors and cashiers), and client modules. Shop modules interact with client modules, so the interface elements of shop modules and of client modules complement each other.
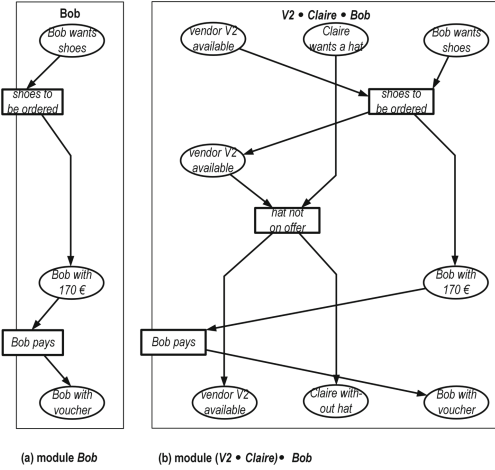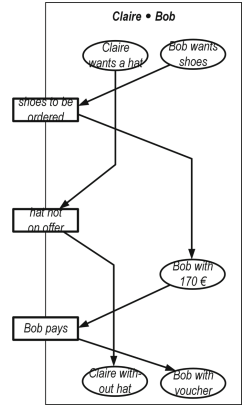
Fig. 5. Composing another module.

Fig. 6. Claire ● Bob.

To continue, Fig. 5 (a) shows the module *Bob* of the client Bob from Fig. 3 (e). As with the *Claire* module, its transitions lie in its left interface. We now compose $V2 ● Claire$ with *Bob* and obtain the module $(V2 ● Claire) ● Bob$ in Fig. 5(b). Alternatively, we could have formed the module *Claire*●*Bob* first (Fig. 6) and then module $V2 ● (Claire ● Bob)$. It is easy to see that the modules $(V2 ● Claire) ● Bob$ and $V2 ● (Claire ● Bob)$ are identical. We will see that in general, the composition operator ● is *associative*.

2.2   The Formal Framework of Modules

As usual, we represent a Petri net as a triple $(P, T; F)$. We employ the usual graphical representation with boxes, circles, and arrows. In this section, we recall a special case of the composition calculus, and particularly occurrence modules and their composition. The general case can be found in [5].

An *interface over* a set Λ *of labels* is a finite set R, with each element of R carrying a label of Λ. We refrain from the general case of two or more equally labeled interface elements here.

For two interfaces R and S, equally labeled elements $r \in R$ and $s \in S$, are a *harmonic pair* of R and S. A harmonic pair is labeled by the label of r and s. The element s is a *harmonic partner* of r in S, and r is a *harmonic partner* of s in R.

A *module* is a Petri net $N = (P, T; F)$ together with two interfaces $^*N$ and $N^* \subseteq P \cup T$, denoted as the *left* and the *right interface* of N. Nodes not in an interface belong to the *interior of N*.

In graphical representations, the interior of N is surrounded by a box, with the elements of the left and the right interface on its left and the right margin, respectively, e.g. Fig. 4.

We are now prepared for the fundamental definition of *composing* two modules:

Let A and B be two modules. For each node x of A or of B, let $x' = \{x, y\}$ if $\{x, y\}$ is a harmonic pair of A^* and $^*B$; let $x' = x$ if no harmonic pair of A^* and $^*B$ contains x. Then the module $A \bullet B$ is defined as follows (each element retains its label):

1. The nodes of $A \bullet B$ are all x' such that x is a node of A or of B.
2. The edges of $A \bullet B$ are all (x', z'), such that (x, z) is an edge of A or of B.
3. The left interface $^*(A \bullet B)$:
   (a) $^*A \subseteq {}^*(A \bullet B)$;
   (b) For $x \in {}^*B$ holds: $x \in {}^*(A \bullet B)$, if x has no harmonic partner in A^*.
4. The right interface $(A \bullet B)^*$:
   (a) $B^* \subseteq (A \bullet B)^*$;
   (b) For $x \in A^*$ holds: $x \in (A \bullet B)^*$, if x has no harmonic partner in $^*B$.

Figures 4, 5 etc. show compositions of modules. Notice that, according to this definition, $^*(A \bullet B)$ or $(A \bullet B)^*$ may acquire different elements with equal labels. However, this never happens in this paper's examples; further details can be found in [5].

A fundamental property of composition is *associativity*, decisive for the usability of modules and their composition. In fact, for any three modules A, B and C holds:

$$(A \bullet B) \bullet C = A \bullet (B \bullet C). \tag{1}$$

As a consequence, it makes sense to just write $A \bullet B \bullet C$. This property is a special case of a more general notion of modules and their associative composition, as discussed in [11].

Furthermore, there are clear criteria for the case of commutativity: for modules A and B holds

$$A \bullet B = B \bullet A \tag{2}$$

if and only if no label occurs in $^*A \cup A^*$ as well as in $^*B \cup B^*$.

The nets in the examples of Sect. 2.1 all exhibit a particular structure: The arcs form no cycles, and each place has at most one ingoing and one outgoing arc:

A net $N = (P, T; F)$ is an *occurrence net* if and only if:

1. The transitive closure of F, usually written as F^+, is a strict partial order, viz. irreflexive and transitive, on $P \cup T$. We denote this relation as $<_N$;
2. for each $p \in P$ there exists at most one arc shaped (t, p) and at most one arc shaped (p, t).

A module is an *occurrence module* if and only if the underlying net is an occurrence net.

For two occurrence modules A and B, the composed module $A \bullet B$ is in general not an occurrence module again. Figure 7 shows an example. This example shows that the interior of A and B matters for this problem. Nevertheless, it can be reduced to a problem of the induced order of interface elements:
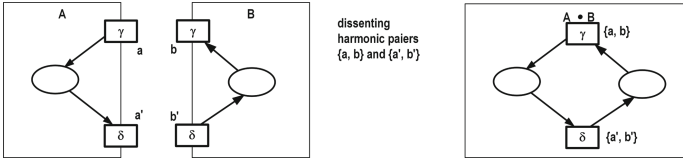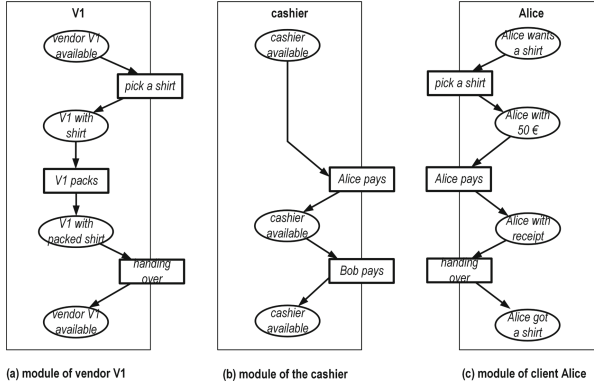
Fig. 7. Dissenting pairs.

Fig. 8. Three further modules.

With a, $a' \in A$ and b, $b' \in B$, let $\{a, b\}$ and $\{a', b'\}$ be harmonic pairs of A^* and $^*B$. They *dissent* if and only if either $a <_A a'$ and $b' <_B b$, or $a' <_A a$ and $b <_B b'$.

Then, for two occurrence modules A and B it holds: $A \bullet B$ is an occurrence module if and only if A^* and $^*B$ have no dissenting harmonic pairs. All compositions of occurrence modules in this paper yields an occurrence module again.

2.3   Completing the Example

We extend the example of Sect. 2.1 by modules for the remaining three agents of Fig. 3, as in Fig. 8. Figure 9 shows compositions of these modules. Interestingly, the module *cashier* • *Alice* in Fig. 9(a) is an example of a module with elements in both the left and right interfaces. Finally, the module in Fig. 9(b) composes all three modules.

We can now compose the composed module in Fig. 9(b) with the composed module in Fig. 5(b), and obtain the composed module

$$V := V1 \bullet cashier \bullet Alice \bullet Bob \bullet V2 \bullet Claire \tag{3}$$

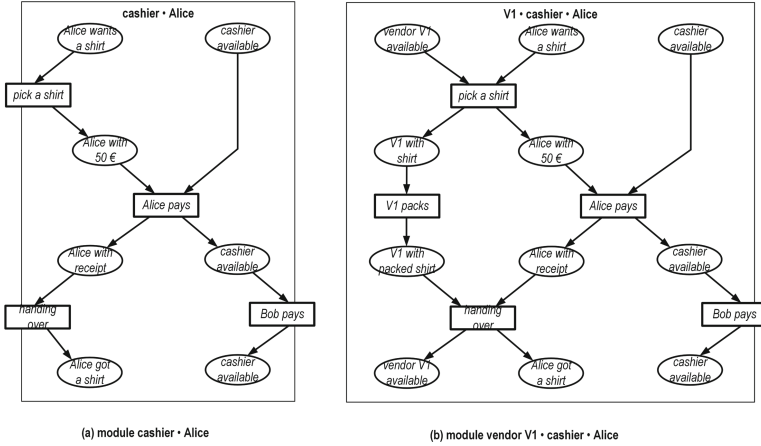in Fig. 10. The two interfaces of this module do not contain any elements.

(a) module cashier • Alice

(b) module vendor V1 • cashier • Alice

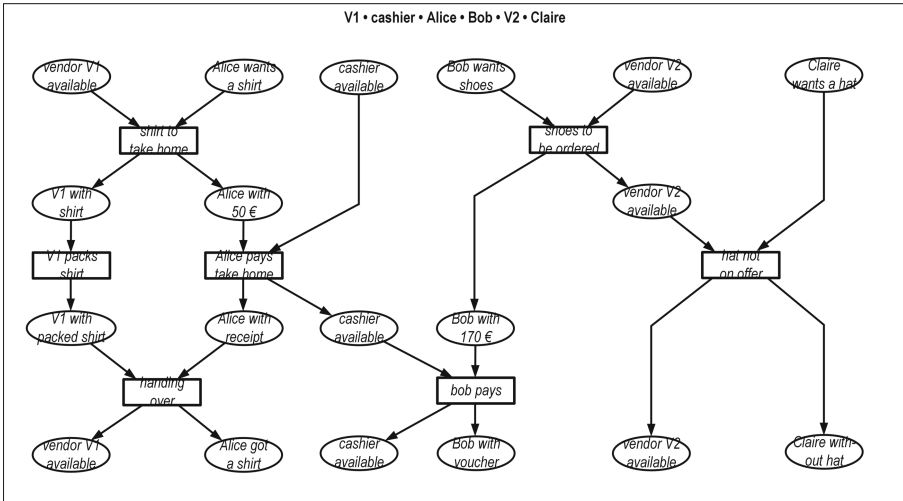**Fig. 9.** Two further module compositions.

Fig. 10. Behavioral module $V := V1 \bullet cashier \bullet Alice \bullet V2 \bullet Claire \bullet Bob.$

From an abstract and more systematic point of view, the expression (3) is a bit unattractive. It would be nicer to have the module $trade := V1 \bullet cashier \bullet V2$, with all interface elements on the right, and the module $customers := Alice \bullet Claire \bullet Bob$ with all interface elements on the left. The module V in (3) is then written as $trade \bullet customers$. Indeed, this is possible without any problems, because the modules $Alice$ and $V2$ have disjoint interfaces. According to equation (2), the sequence of the two modules $V2$ and $Alice$ in (3) can be swapped.

Summing up, the module V of Fig. 10 represents a typical single *run* of a system. V provides insight into subtle details of the mutual relationship of

the events of the joint behavior of the involved six agents. For example, in the presented run, the joint events of the modules $V1$, *Alice* and the *cashier* are detached from the events of the modules of the other three agents. Bob waits until the cashier is finished with Alice. But vendor $V2$ and Alice are not related at all to the cashier.

All this insight has been gained from the event log of Fig. 2, together with the intuitively obvious idea that events of the business people will never be merged, hence they come with elements in right interfaces only, and correspondingly, events of the customers will never be merged, thus all come with elements in left interfaces. The choice of left and right interface is motivated by the dichotomy between shop modules (vendors and cashiers), and client modules. Of course, right and left may be swapped here. So, the interface elements of shop modules and of client modules complement each other.

2.4   Composing an Occurrence Module from Occurrence Atoms

Here we consider an alternative way of constructing occurrence modules. In Sec. 2.3 we composed the run in Fig. 10 from the modules of the six behavioral strands of agents, given in Fig. 3. Occurrence modules are frequently, but not always, composed from modules generated by such agents. Alternatively, an occurrence module can be generated from *occurrence atoms*. An occurrence atom is a module that represents a single transition together with its surrounding arcs and places. We denote the occurrence atom of a transition t by $\underline{t}$. To correspond to the previous representation of occurrence modules, we place the left interface of an occurrence atom at the top and the right interface at the bottom of its graphical representation.

Figure 11(a), (b) and (c) show the occurrence atoms of the transitions *shirt to take home*, *V1 packs shirt*, and *Alice pays take home* of module V of Fig. 10. The composition of the three occurrence atoms in Fig. 12 is identical to the upper left part of module V. It is easy to see how the occurrence atoms of the remaining four transitions of V can be generated, and that their composition yields the entire module V. In fact, this is generally true: the occurrence atoms $\underline{t}$ of the transitions t of an occurrence net N can be arranged as a sequence $\underline{t_1}, \ldots, \underline{t_n}$ such that

$$N = \underline{t_1} \bullet \cdots \bullet \underline{t_n}. \tag{4}$$

This representation will be used in the following sections.

3   System Models

So far, we showed how to deduce a single run from a given event log. Our aim, however, is to deduce a system model from an event log. To this end, we need a manageable kind of system models. Here we derive such system models.
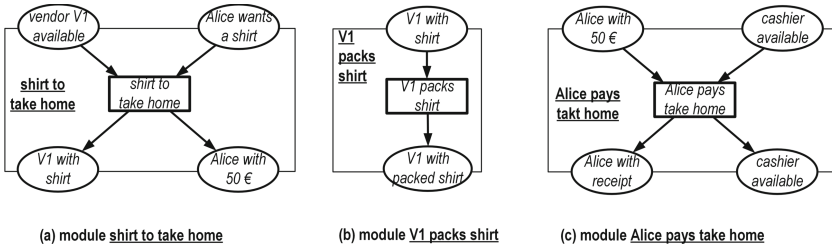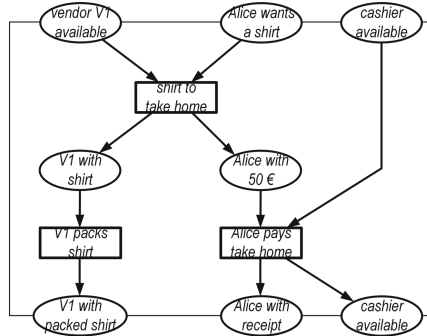
Fig. 11. Occurrence atoms

Fig. 12. Module M0: *shirt to take home ● V1 packs shirt ● Alice pays take home*

Many similar logs would yield many similar runs. Now we show how to extrude a system model from a set of runs. In a first step, we concentrate on the systematic management of involved data and functions.

3.1   Structures and Signatures

To cope with data and functions on data, we employ *signatures* and *signature-structures*, well-known in mathematics from general algebra and first order logic, and in informatics from algebraic specifications [12]. Figure 13 shows the signature-structure S_0 for the running example, consisting of eight sets and four functions. Each set is finite and includes real or imagined objects such as *clients*, *vendors*, *cashiers*, *products*, *vouchers*, *wrapped items*, but also more abstract items such as *money* and *descriptions of items*. In the course of systems mining, a structure like this should be provided by the provider of the logs. It may also be deducible from the logs.

A symbolic representation of a system requires abstract, symbolic representations of structures such as in Fig. 13. This is achieved by means of signatures: a *signature* Σ_0 for a structure S includes *sorted symbols*: a symbol for each set and each function of S. Figure 14 gives a signature, Σ_0, for the above structure S_0. For the sake of simple notation, for each set and each function of S_0 we write the corresponding symbol of Σ_0 in italic.

sets
M: {Alice, Bob, Clara} *set of clients*
V: {C1, C2} *set of vendors*
C: {c1} *set of cashiers*
I: {hat, shoes, shirt} *set of items*

P: {1€, 2€, ... , 1000€} *amounts of money*
V: {<hat>, <shoes>, <shirt>} *set of vouchers*
W: {[hat], [shoes], [shirt]} *set of wrapped items*
I: {hat, shoes, shirt} *set of descriptions of items*

functions
f: I → P
f(hat) = 100€   **f(shoes) = 200€**
                  f(shirt) = 50€

p: I → W
p(z) = [z] for z ∈ I.

v: I → V
v(z) = <z> for z ∈ I.

_: I → I
_(z) = z

set symbols	function symbols	variables
M: clients	*f: I → P*	**x:** *clients*
V: vendors	*p: I → W*	**y:** *vendors*
C: cashiers	*v: I → V*	**z:** *items*
I: items	*_: I → I*	**z:** description
P: prices		of item **z**
V: vouchers		**c:** cashier
W: wrapped items		
I: descriptions of items		

Fig. 13. Structure S_0, describing the system's data.

Fig. 14. Signature Σ_0 and variables for the structure S_0

Additionally, Fig. 14 shows sorted variables. Sorted symbols and variables yield *terms*, such as $f(z)$, or tuples of terms, such as $(x, f(z))$. A *valuation* β of the variables assigns to each variable v an item $\beta(v)$ of the structure S_0. For example, with $\beta(x) = Alice$, $\beta(y) = V1$ and $\beta(z) = shirt$, the tuples (y, z) and $(x, f(z))$ yield in S_0 the tuples

$$\beta(y, z) = (V1, shirt) \text{ and } \beta(x, f(z)) = (Alice, 50€) \tag{5}$$

3.2 System Atoms and Their Composition

In order to extrude a system model from a set of runs, we start from single occurrence atoms, extruding a more general model of system atoms. Figure 15 shows an example: The atom *shirt to take home* of Fig. 11(a) (repeated in Fig. 15(a)) is re-written in Fig. 15(b): information about the vendor V1, the client Alice, the item shirt, and the price 50 € moves from the module's places to its arcs. This representation is now conceived as an instantiation of the *item to take home* module in Fig. 15(c). In this module, the constant arc inscriptions of Fig. 15(b) are replaced by the variables x, y, and z, and terms $f(z)$ and z. In Figs. 15(b) and (c), the place inscriptions of the left (upper) interface places are conceived as tokens of the Petri net. Then, the firing rule of Petri nets defines the tokens for the right (lower) interface places. Figure 15(b) is now gained as the instantiation of Fig. 15(c) by means of the above valuation β as in (5). Of course, different valuations yield different instantiations of the *item to take home* system atom. This way, Fig. 15(c) is a system atom, representing many occurrence atoms.

In analogy to Fig. 15(a), (b), and (c), Fig. 15(d), (e), and (f), generalizes the *V1 packs shirt* occurrence atom as in Fig. 11(b). It is obvious how from the five remaining occurrence atoms, the corresponding system atoms can be deduced.

Figure 16 composes the seven system atoms. This is a *symbolic occurrence module*. Content wise, with the valuation β for all arc inscriptions, it is just a different representation of the occurrence module V in Fig. 10. Denotations of places and transitions have slightly been adjusted to better support intuition. The place inscriptions of V are gained in Fig. 16 by the Petri net firing rule. Further, Fig. 16 has a non-empty left and right interface, in contrast to Fig. 10.
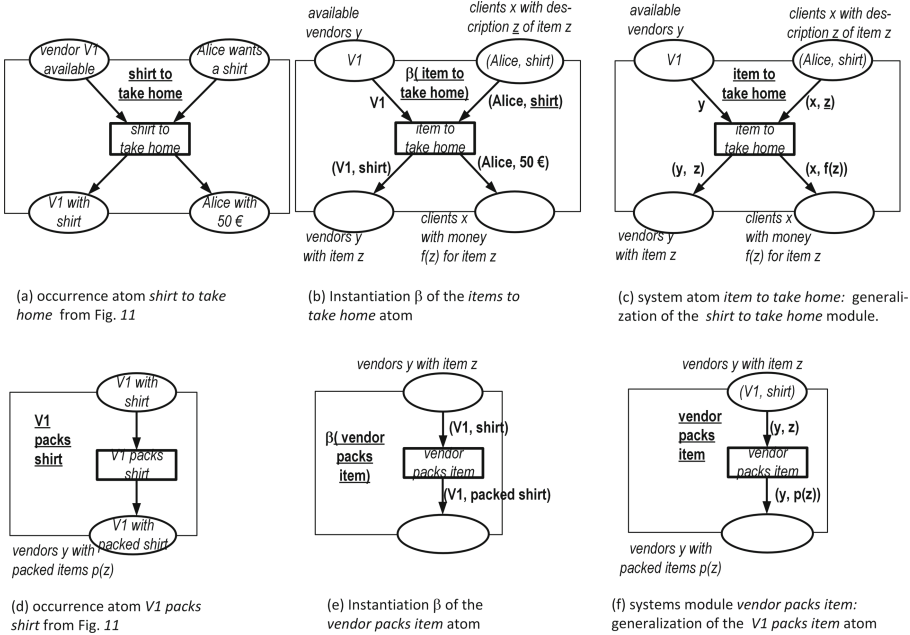
(a) occurrence atom *shirt to take home* from Fig. *11*

(b) Instantiation β of the *items to take home* atom

(c) system atom *item to take home:* generalization of the *shirt to take home* module.

(d) occurrence atom *V1 packs shirt* from Fig. *11*

(e) Instantiation β of the *vendor packs item* atom

(f) systems module *vendor packs item:* generalization of the *V1 packs item* atom

Fig. 15. Occurrence atoms and system atoms

3.3   Constructing a System Net from Symbolic Occurrence Modules

It is now possible to deduce a full-fledged Petri net model from the symbolic occurrence module of Fig. 16: Just identify equally labeled places. The resulting system net is shown in Fig. 17. The tokens of Fig. 16 are collected on the corresponding place of Fig. 17.

Figure 17 shows a high level Petri net. It specifies a lot of runs, depending on the choice of the valuation β of the variables. Furthermore, now, even when fixing the valuation β as above, each client with his description of an item may now execute any of the three events *item not on offer*, *item to be ordered* and *item to take home*, with any of the two vendors $V1$ or $V2$. This is a generalization that suggests itself from the assumptions of the system.

3.4   Deriving a Net Schema

The system model in Fig. 17 fixes the sets of vendors, clients, and items. One would prefer a specification that leaves these sets open, to be fixed as an *interpretation* of those symbols by the user of the model. For this purpose, it suggests itself to use fresh symbols, e.g. *VE*, *CL*, and *CA*, to be interpreted as sets of vendors, clients, and cashiers, as initial tokens on the places *available vendors*, *clients with descriptions of items*, and *available cashiers*, resp. However, this is not exactly what we want: An interpretation of *VE* would, for example, interpret
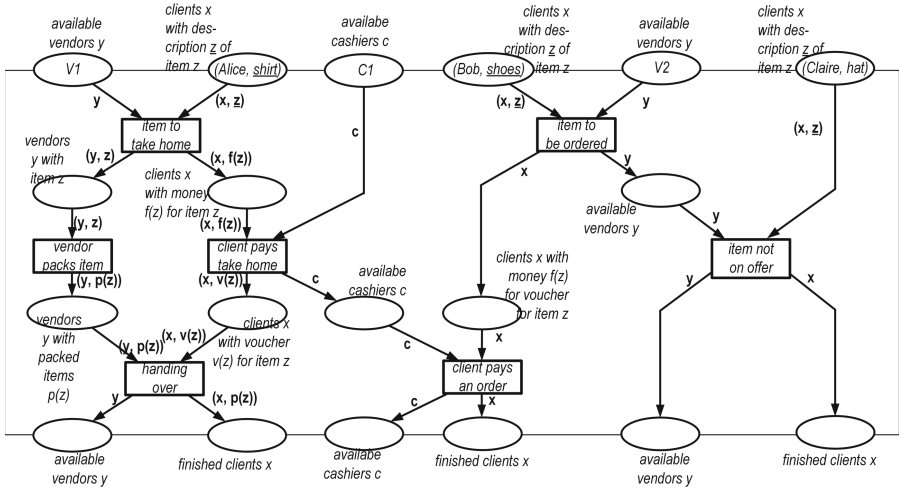
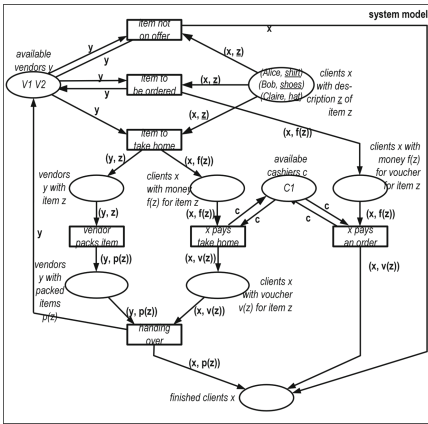**Fig. 16.** Composed atoms: symbolic representation of the behavioral module V.
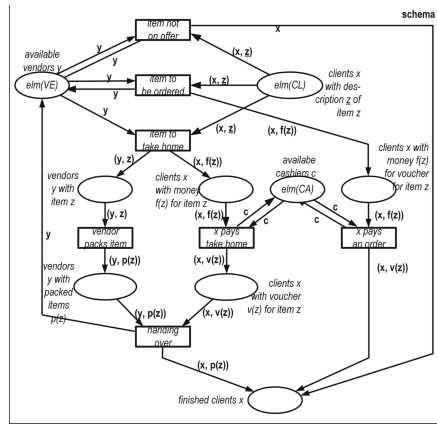

Fig. 17. System model M.

Fig. 18. Schema: symbolic initial marking.

the symbol VE by the set $\{V1, V2\}$ as one token on the place *available vendors*. Instead, we want two tokens, $V1$ and $V2$. This is represented by means of the "elm"-notation, as in the net of Fig. 18 (more details in [5]).

4   How to Mine a System Model

The notions, concepts, and constructs described in the above sections suggest to mine a system model starting from information on static system aspects such as the architecture of the system, the data structures, and the involved agents.

The data and the operations on the data are systematically represented in a signature-structure. The architecture and the agents provide the background for modeling dynamic aspects, i.e. for the derivation of occurrence modules as models for runs, and finally system modules as models for the entire system.

4.1   From Logs to Runs

The first step identifies for each agent its sequential behavior from the log, and constructs a distributed run from the agents' behaviors:

1. From a given event log, for each agent identify in the log the events which involve the agent. The sequence of those events constitute the *behavior* of the agent in the log. Figures 2 and 3 show corresponding examples.
2. Turn the behavior of each agent into an occurrence module: each event either belongs to the module's interior part, or its left or its right interface. For an element, adequate choice of the interface depends on the intended composition with elements of other modules. Figures 4(a), and (b), 5(a), and 8(a), (b), and (c) show examples.
3. Compose the agents' occurrence modules: In general, an event of an event log is involved in more than one agents' behavioral module. Composition of the modules yields a comprehensive occurrence module, i.e. partially ordered run, as in Fig. 10.

4.2   From Runs to Systems

The second step identifies for each occurrence atom of a given partially ordered run a system atom with terms over the given signature-structure as arc inscriptions. From this representation, the sought system model is derived:

4. For each occurrence atom of the run, identify the involved agents and data structures. Move this information from place inscriptions to arc inscriptions. The arc inscriptions then are terms of the underlying signature structure. Figure 15 shows examples.
5. In this representation of each occurrence atom, replace each constant symbol by a variable. This yields a system atom.
6. Compose those system atoms, as in Fig. 16.
7. In this representation, merge equally denoted places. This yields the sought system model, as in Fig. 17.
8. To achieve a purely schematic representation, replace the initial marking by a symbolic marking, as in Fig. 18.

5   Related Work

The main concepts for the theoretical foundations of process mining are based on the idea of grammar inference, grammar induction, or language identification [9], which was originally proposed by [8]. Since these theoretical models do not adequately represent all interesting aspects of business processes,

a plethora of enhanced formal frameworks are developed [1]. However, none of these approaches are completely satisfactory because the role of causality, subsystems, and data are not integrated and adequately covered. Besides the theoretical work, many practical approaches originate from engineering process modeling and mining systems [7]. However, these approaches lack a theoretical foundation.

Recent work in the area of artifact-centric [4], object-centric process mining [2], and agent system mining [13], addresses these lacks already. Although these ideas clearly show improvements compared to the classical understanding of systems, models, and logs as formal languages, they still do not provide a satisfactory understanding of system architecture and the difference of abstract and concrete data structures which are strongly needed for an integrated understanding of business systems. Additionally, our understanding of an agent is rather general compared to the technical notion used by [13].

Although recent work acknowledges the need for representing causal structures, the choice is often not satisfactory. C.A. Petri formulated the concept of distributed runs as early as the late 1970s [10]. It has been taken up again and again, also under the names "true concurrency", or "partial order semantics", but initially did not prevail over sequential processes. One of the reasons for this was the comparatively complex technical apparatus for dealing with distributed processes, combined with comparatively little benefit. Meanwhile, the basic ideas of distributed runs are used in many contexts, e.g. partial order process mining [14]. Furthermore, the composition calculus, as used in this contribution, provides adequate and simple technical tools.

To cope properly with data aspects, and in particular to properly integrate behavioral and data aspects in one formal framework, we resort to signature-structures, the established formal basis of first order logic and algebraic specifications [12]. Models of really big systems are gained by composing models of subsystems. The composition calculus covers also this aspect, as developed in [5].

6   Conclusion

Classical process mining assumes a run as a sequence of events and then tries to solicit information about concurrent, independent event occurrences a and b form the observation, that in many similar logs, a and b occur in either order. We suggest to start considering a run as an unordered set of events, and then to order them, as much as reasonable, by considering agents and the composition of agents' behavior. For example, the module V of Fig. 10 provides insight into subtle details of the mutual relationship of the events of the joint behavior of the involved six agents. In the presented run, the joint events of the modules *vendor V1*, *Alice* and *cashier* are detached from the events of the modules of the other three agents. Bob waits until the cashier is finished with Alice. But vendor $V2$ and Alice are not related at all to the cashier. All this has been gained from the event log of Fig. 2, together with the intuitively obvious idea

that the events of the trade components never should be merged, hence all go to the right interfaces, and correspondingly the events of the customers should never be merged, thus all go to the left interfaces. Of course, right and left may be swapped here.

In this paper, we argue that *causality, composition*, and *objects matter* while mining a system. We introduce the foundational concepts for conducting system mining. In the future, more case studies need to be done and new tools for supporting the main ideas of HERAKLIT have to be developed. So, in the future, we speculate that the two academic worlds of data and process mining will be complemented with and enhanced by systems mining allowing a deeply integrated understanding of business processes.

References

1. van der Aalst, W.: Process Mining: Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. van der Aalst, W., Berti, A.: Discovering object-centric petri nets. Fund. Inform. **175**(1–4), 1–40 (2020)
3. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.: Quality dimensions in process discovery: the importance of fitness, precision, generalization and simplicity. Int. J. Cooper. Inf. Syst. **23**(1), 1440001/1-39 (2014)
4. Fahland, D.: Artifact-centric process mining. In: Sakr, S., Zomaya, A.Y. (eds.) Encyclopedia of Big Data Technologies, pp. 108–117. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-77525-8_93
5. Fettke, P., Reisig, W.: Handbook of HERAKLIT (2021). HERAKLIT working paper, v1.1, 20 September 2021. http://www.heraklit.org
6. Fettke, P., Reisig, W.: Modelling service-oriented systems and cloud services with HERAKLIT. In: Zirpins, C., et al. (eds.) ESOCC 2020. CCIS, vol. 1360, pp. 77–89. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-71906-7_7
7. Gartner: Market guide for process mining. Technical report ID G00387812 (2019)
8. Gold, E.M.: Language identification in the limit. Inf. Control **10**(5), 447–474 (1967)
9. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, Cambridge (2010)
10. Petri, C.A.: Non-sequential processes. Technical report ISF-77-5, Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin, Federal Republic of Germany (1977)
11. Reisig, W.: Associative composition of components with double-sided interfaces. Acta Informatica **56**(3), 229–253 (2018). https://doi.org/10.1007/s00236-018-0328-7
12. Sanella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-17336-3
13. Tour, A., Polyvyanyy, A., Kalenkova, A.: Agent system mining: vision, benefits, and challenges. IEEE Access **9**, 99480–99494 (2021)
14. van der Aa, H., Leopold, H., Weidlich, M.: Partial order resolution of event logs for process conformance checking. Decis. Support Syst. **136**, 113347 (2020)