HERAKLIT case study: retailer

Peter Fettke[1,2][0000−0002−0624−4431] and Wolfgang Reisig[3][0000−0002−7026−2810]

[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. The modeling method HERAKLIT is presented through a business case study. The case study demonstrates how HERAKLIT can be used to systematically model, abstract, compose and analyze computer-integrated systems of business practice. Structural aspects, abstract descriptions of data and objects, and behavioral models are integrated in a novel way.

Keywords: systems composition · data modeling · behavior modeling · process modeling · composition calculus · algebraic specification · Petri nets · Systems Mining

Preliminary remarks

What is HERAKLIT?

HERAKLIT is a method for modeling computer-integrated business systems. The objective of HERAKLIT is outlined by the HERAKLIT *memorandum* [4]. The systematic structure of HERAKLIT as well as the selection of the individual concepts and their interaction are motivated by the HERAKLIT *manual*, which is not yet available. The HERAKLIT *tool* supports the modeler in creating and analyzing descriptive models, particularly extensive models that use overviews and abstractions in addition to detailed representations of behavior. This tool will be made available in the future.

What does this text offer?

For readers without previous knowledge, a comprehensive case study explains how to construct HERAKLIT models. We model the central business processes of a retailer and its business network partners, we show how they can be composed and abstracted, and we describe their behavior. From the constructs of this case study the reader can intuitively and easily understand the general concepts of HERAKLIT.

cite as: FETTKE, P.; REISIG, W.: HERAKLIT *case study: retailer*. 2020. – HERAKLIT working paper, v1, December 21, 2020, http://www.heraklit.org

How is the text structured?

The case study is divided into eight parts. Part I introduces the content of the case study and the developed model. A HERAKLIT model is basically divided into individual modules which are presented in part II. Part III explains how data and objects are modeled in the case study. The concept of distributed runs is introduced in part IV. The introduced concepts are transferred to schemata to talk about arbitrary data, objects and runs in part V. Part VI then details the individual modules in terms of their structure and behavior. The complete model overview is shown in part VII, while part VIII explains how further views on the complete model can be described.

I Content of the case study and the developed model

1 The retailer of the case study

A retailer sells articles to its customers via an online shop. No articles are sold via other forms of distribution such as stationary retail, wholesale or agents.

The retailer has three types of business network partners:

- its *customers*: they send in orders; the retailer informs the customers about delivery dates, et cetera;
- the *supplier*: the retailer reorders out-of-stock goods; the supplier delivers ordered goods to the retailer;
- the *freight forwarders*: they receive packed freight from the retailer and deliver it to the customers.

The retailer itself is divided into three parts:

- The *order management* receives orders from customers, asks the inventory management to confirm the availability of each ordered item and issues delivery orders to the warehouse. If necessary, the order management system initiates partial deliveries.
- *Inventory management* keeps a list of the inventory in the warehouse. If an item's available quantity becomes too low, inventory management reorders from the supplier and asks the warehouse to acknowledge the receipt of these items.
- The *warehouse* packs ordered goods according to the delivery orders of the order management and hands over the freight to a freight forwarder. It also receives goods from the supplier, sorts them into the warehouse and notifies the inventory management system that the goods have been received.

The retailer and its partners form a system that must function correctly as a whole. For example, all ordered items should be delivered to a customer if necessary, in partial deliveries. Conversely, only those articles that the customer has ordered should be delivered. Within the retailer, inventory management should only prompt the warehouse to deliver if the necessary articles are available in the warehouse.

2 The HERAKLIT model for the retailer and its business network partners

We construct a HERAKLIT model for the entire system, containing the described retailer and its three business network partners. First we model the abstract structure of the whole system and the three business network partners of the retailer as HERAKLIT *modules* in the context of their principal interaction. This is followed by a detailed and systematic presentation of the data and objects in an abstract version. The representation on a schematic level is left to concrete *instantiations* which specify how exactly customers, articles, goods, freight, et cetera look. Finally, the behavior of the individual modules is represented in the form of local steps and mathematical operations on the data and objects involved.

The model leaves a number of decisions open. In particular, order management

- can bundle available items in the warehouse into partial deliveries according to particular choices;
- can select different goods in the warehouse for the same ordered item;
- does not determine which of the available freight forwarders will deliver.

Thus, the model precisely captures the aspects of trading operations described in Section 1.

The model combines principles known and proven from *algebraic specifications*, *Petri nets* and the *composition calculus*. They are intuitively obvious, so that no previous knowledge is required to understand the graphical model. At the same time, the model is formal; thus the above-mentioned properties can be formally formulated and proven in the model. Details will be given in the HERAKLIT manual.

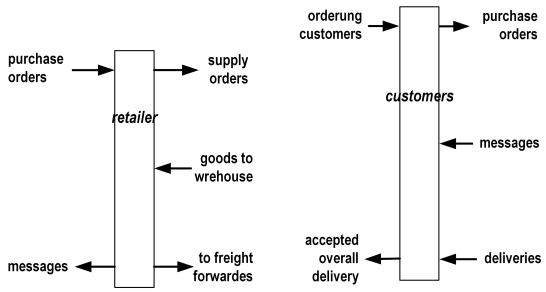
II The concept of a module

3 HERAKLIT modules

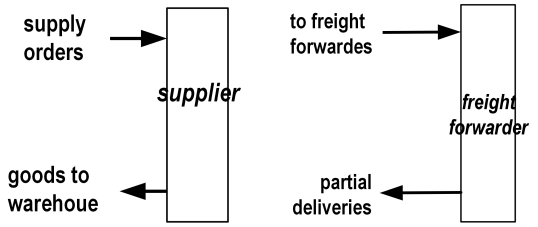
A real system generally consists of subsystems that are composed. The central modeling concept is based on this obvious observation: A HERAKLIT module is a model, graphically represented as a rectangle, with two essential aspects:

- Its *interior*: this can be freely chosen. It may consist only of the name of the module, or show its structure or its dynamic behavior; and
- Its *surface*: this consists of a set of *gates*. Each gate has a *label*, which serves as a caption. Each gate is represented graphically as a short line that starts at the rectangle of the component and ends at the label of the gate.

The module in figure 1a shows how this works in detail, using the example of the retailer: The interior of the retailer is shown inside the rectangle; here we abstract completely from internal details and only write the name of the



(a) the retailer (b) the customers



(c) the supplier (d) the freight forwarders

Fig. 1: abstract HERAKLIT modules: the retailer and its three kinds of partners

module. The interface consists of five gates with the labels *purchase orders* (from customers), *messages* (to customers), *supplier orders* (to the supplier), *delivered goods* (from the supplier), and *freight delivery* (to freight forwarders).

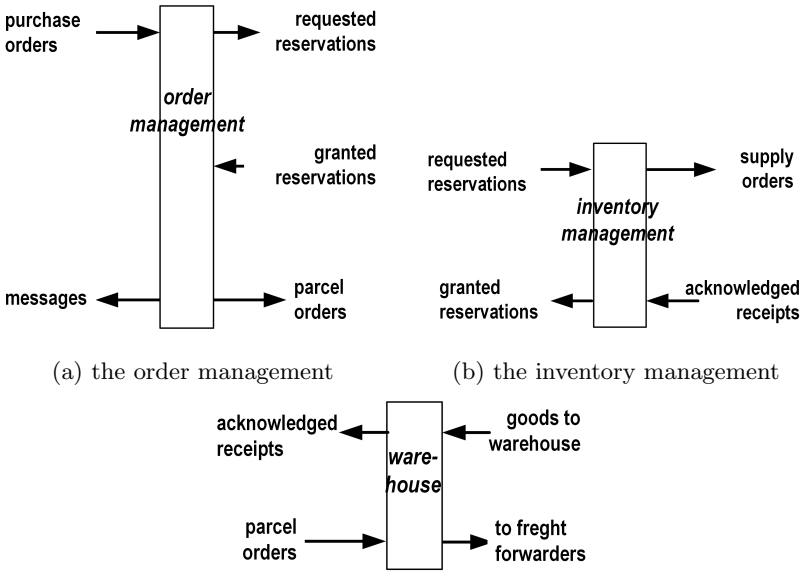
Similarly, the modules of the figures 1b, 1c and 1d show the three types of business network partner: *customer*, *supplier* and *freight forwarder* with their respective gates. Figure 2 shows the three departments of the retailer, again as HERAKLIT modules.

All HERAKLIT modules in figures 1 and 2 are *abstract* modules: Their interior contains only the name of the module. Typically, however, the interior of a HERAKLIT module shows details of its structure or behavior. The structure of a module is often a *composition* of submodules.

Each gate of the modules in figures 1 and 2 is – intuitively formulated – an *entrance* or an *exit* through which objects, documents, information, et cetera flow. The direction of this flow helps to understand the behavior of the module; we indicate it with an arrowhead. Such arrowheads – and other labels and captions – are not part of the formal framework of HERAKLIT.

4 The composition of HERAKLIT modules

Subsystems of a large real system are reasonably formed in such a way that they can be composed according to a fixed procedure and their composition returns the original system. HERAKLIT follows this concept also for the composition



(a) the order management (b) the inventory management

(c) the warehouse

Fig. 2: abstract versions of the three departments of the retailer

of HERAKLIT modules. Technically, the composition $L \bullet M$ of two HERAKLIT modules L and M is defined as a merger of equally labeled gates of L and M. The merged element then remains inside $L \bullet M$. The other gates of L and M become gates of $L \bullet M$.

Figure 3a shows an example: The *order management* and the *inventory management* from figures 2a and 2b both have gates with the label *requested reservations*. In figure 3a the two gates merge to one element inside *order management* • *inventory management*. The same applies to the two gates with the label *confirmation of availability*. The other three gates of *order management* and the other two gates of *inventory management* then form the interface of the composed system *order management* • *inventory management*.

Figure 3b depicts the system

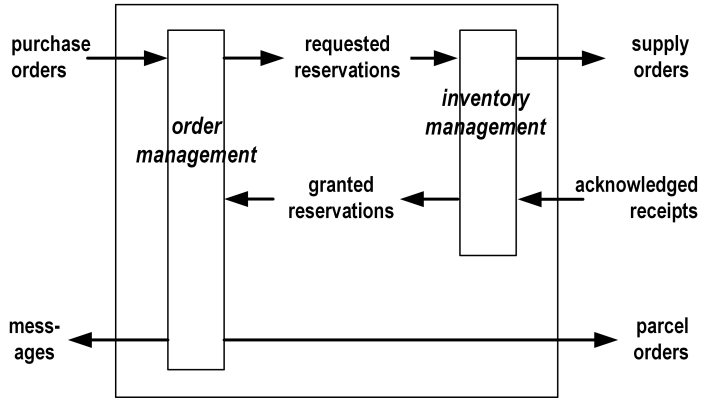
$$\textit{inventory management} \bullet \textit{warehouse}.$$

Both composed modules can now be extended by the missing third department of the retailer. As a result, the following two modules can be composed:

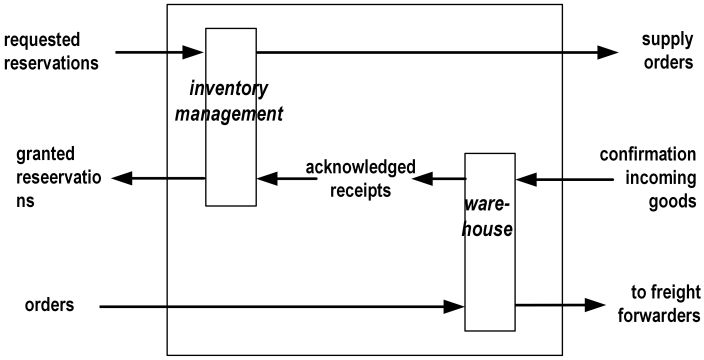
$$(\textit{order management} \bullet \textit{inventory management}) \bullet \textit{warehouse}$$

and

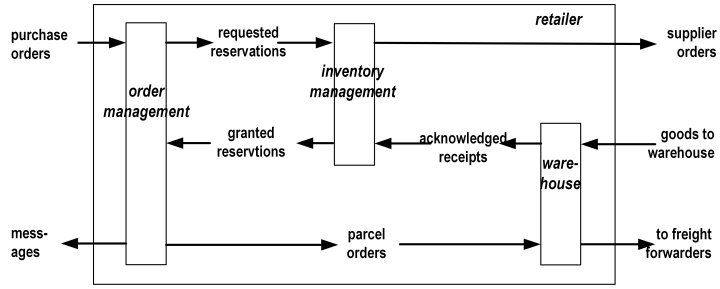
$$\textit{order management} \bullet (\textit{inventory management} \bullet \textit{warehouse}).$$



(a) the composition *order management* • *inventory management*



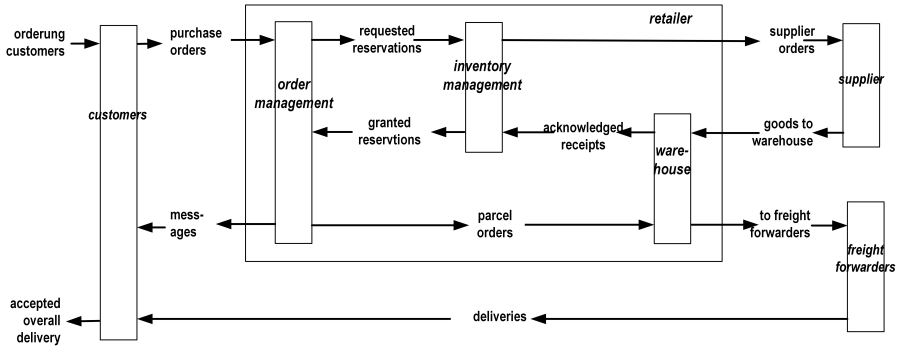
(b) the module *inventory management* • *warehouse*



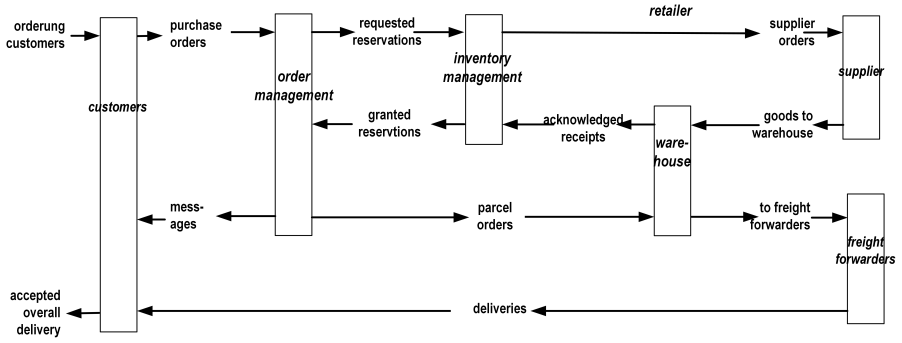
(c) the retailer, defined as
order management • *inventory management* • *stock*

Fig. 3: composition of departments

These two modules are identical; therefore, in figure 3c the brackets can be omitted.



(a) the composition $customers \bullet retailer \bullet supplier \bullet freightforwarder$



(b) the composition $customers \bullet order\,management \bullet inventory\,management \bullet warehouse \bullet supplier \bullet freight\,forwarder$

Fig. 4: the retailer in the context of its business network partners

Some details of the definition of composition are described in the appendix and particularly in the HERAKLIT manual. Two properties of the HERAKLIT composition operator are central: It is

- *total*, so any two HERAKLIT modules L and M can be composed to the HERAKLIT module $L \bullet M$, and it is
- *associative*, so for any module L, M and N: $(L \bullet M) \bullet N = L \bullet (M \bullet N)$; therefore, you never have to parenthesize.

Both properties are fundamentally important for a useful composition operator for modules and thus for models of real systems.

Figure 4a composes the retailer module with the modules for the customers, the suppliers and the freight forwarders to the *overall module* of the case study. The modules involved are then *submodules* of the overall module.

Each gate of the retailer module is by design also a gate of one of its three submodules. Therefore, in figure 4a the retailer module can be replaced by its

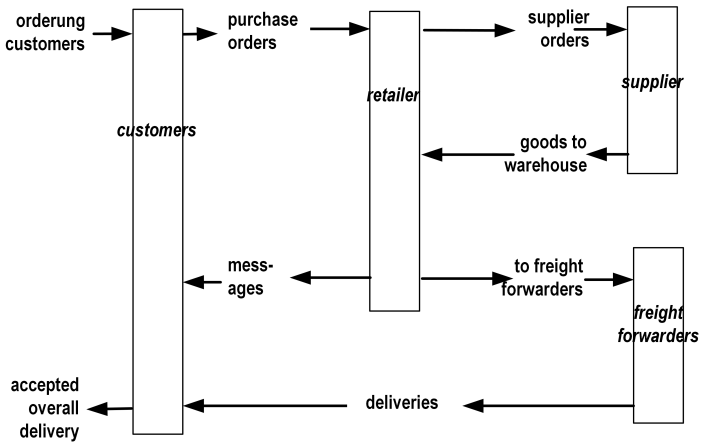


Fig. 5: Composition of abstractions: [customers] • [retailer] • [supplier] • [freight forwarders]

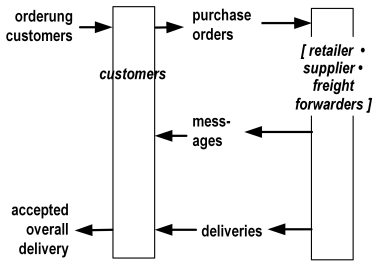
three submodules, as in figure 4b. As explained at the end of section 3, an arrowhead at a gate intuitively indicates a flow direction. Such arrowheads are also intuitively useful in composed modules.

5 Abstraction of HERAKLIT modules

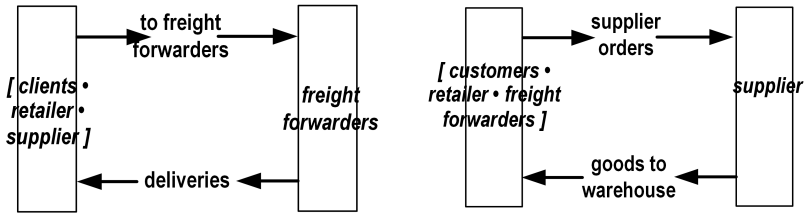
The module in figure 3c is not abstract, but has an inner structure. If one abstracts from this inner structure, one obtains figure 3a, the *abstract version* of the retailer. In general, each module has its uniquely determined *abstract version*: The surface remains the same, but the interior is deleted except for the name of the module. For a module L, $[L]$ denotes its abstract version. Figure 3c is named "retailer"; therefore – strictly speaking – figure 1a must be named "[*retailer*]". But now one can see that the modules in figure 1 are all abstract; therefore, the representation remains unique even without the abstraction operator $[\cdot]$. If one forms a module as a composition of given modules and would like to refer to the abstract version, the abstraction operator removes ambiguity. An example is shown in the comparison of figure 4a with figure 5.

The transition from a module L to its abstract version $[L]$ can be reversed and L can be seen as one of many possible refinements of $[L]$. For example, figure 3c is a refinement of figure 1a. If in a composed module $L = L_1 \bullet L_2 \bullet ... \bullet L_n$ a submodule L_i is refined or abstracted, nothing changes in the other submodules.

The three submodules depicted in figure 3c will be further refined later, as will the business network partners given in figure 1.



(a) the system from the customer's perspective: $customers \bullet [retailer \bullet supplier \bullet freight\ forwarder]$



(b) the system from the perspective of the freight forwarders: $[customers \bullet retailer \bullet supplier] \bullet freight\ forwarders$

(c) the system from the suppliers' perspective: $[customers \bullet retailer \bullet freight\ forwarders] \bullet supplier$

Fig. 6: the system from the perspective of the three business network partners

6 Views

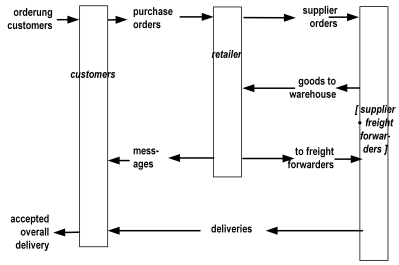
Abstraction and composition of submodules result in different views of the overall system. In particular, each business network partner has its own view, namely the abstraction of the other composed submodules. Figure 6 depicts these three views.

A module (at any level of detail) "sees" the rest of the system generally in two parts: the submodule on its left and the one on its right. Figure 7a shows this perspective of the retailer. But each of its submodules also has its own perspective, as shown in figures 7b, 7c and 7d.

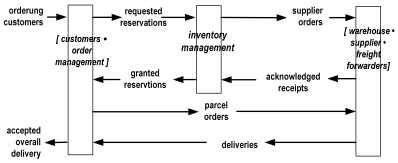
III Data and objects in HERAKLIT models

7 Data, things, types, multisets

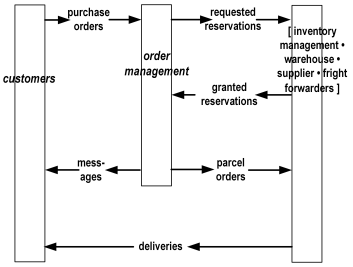
In order to show the expressiveness of HERAKLIT, we examine some data aspects of the case study in more detail. An *order* consists of a *customer* (more precisely: a customer name) and an *article list*. This in turn is a set of *items*. And each item consists of an *article* and the *number* of ordered units.



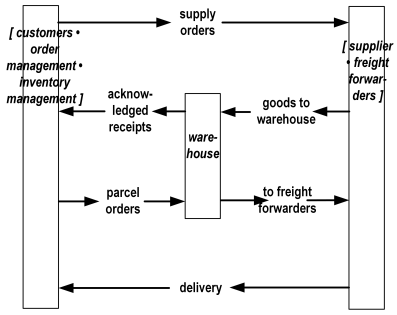
(a) perspective of the retailer: $[customers] \bullet retailer \bullet [supplier \bullet freight\ forwarders]$



(b) perspective of inventory management: $[customers \bullet order\ management] \bullet inventory\ management \bullet [warehouse \bullet freight\ forwarders]$



(c) perspective of the order management: $[customers] \bullet order, management \bullet [inventory\ management \bullet warehouse \bullet supplier \bullet freight forwarders]$



(d) perspective of the warehouse: $[customer \bullet order\ management \bullet inventory\ management] \bullet warehouse \bullet [supplier \bullet freight\ forwarders]$

Fig. 7: perspectives of the retailer and its departments

Customer names, articles, article positions and article lists are *data*; one can display them in a catalog, send them digitally, print them on paper, et cetera. In particular, they can be processed in digital form by a computer.

In addition, there are concrete, real-world *things*, in the example the *shoes, pants, shirts* and *hats*, that are the goods that the supplier sends to the warehouse, which the warehouse packs and hands over to the freight forwarders as freight, which is then delivered to the customers.

Data and things have very different properties: A data element such as an article list can be easily copied or disassembled; an object cannot. One can do little more with an object than bundle it together with others in freight, transport it from place to place and then open the bundle again. From this follows an interesting property that data does not have: An existing thing, for

example a hat, is always in exactly one place. Data and things are *objects*, which HERAKLIT represents in the same formalism.

In the systematic structure of HERAKLIT, objects are elements of *sets*. They are, therefore, not to be understood in the special sense of object-oriented modeling or programming. Often objects are *multisets*. In a multiset, an element can occur more than once: for example, a delivery for an order can contain two hats and three pairs of shoes without distinguishing between the two hats or the pairs of shoes. We write this order as multiset $[hat, hat, shoes, shoes, shoes]$. Formally, a multiset with elements from a set A is a mapping $M : A \to \mathbb{N}$, which assigns each element of A its number of occurrences in M; in the example, $M(hat) = 2$ and $M(shoes) = 3$.

Multisets L and M of a set A can be added:

$$(L + M)(a) \coloneqq L(a) + M(a),$$

multiplied by a scalar (a natural number n)

$$(n \cdot M)(a) \coloneqq n \cdot (M(a)),$$

and compared with each other:

$$L \leqslant M \coloneqq \text{for all } a \in A : L(a) \leqslant M(a).$$

The power set $P(M)$ is the set of all sets N with $N \leqslant M$. $M(A)$ is the set of all multisets over A.

Finally, we use *predicates*: A predicate p either applies to an element, or it does not. For a set M we write $p(elm(M))$ if p applies to every element of M.

Figure 8 summarizes the notations used in this document.

8 The structure S and the schema of the retailer

The model of the retailer consists of four different types of objects:

- *basis sorts*: these are *customers*, *articles*, *dates*, *goods*, *freight forwarders*;
- *derived sorts*: an *article position* consists of an article and a quantity, an *article list* is a multiset of article positions, an *article set* is a multiset of articles, and a *goods set* is a multiset of goods;
- *constants* that explicitly appear in the model: specific items, customers, and initially listed items, goods, and freight forwarders;
- *functions*, which relate basic and derived sorts to each other: Each article position p and each article list a corresponds to a multiset p and a of articles, respectively, and each good w corresponds to an article $f(w)$;
- *predicates* are derived from the basics: *ordering customers*, *sent orders*, *copies of sent orders*, *delivered goods*, et cetera are predicates. A predicate either applies or does not apply to an element of a set. Example: The predicate *sent orders* applies to all orders that have been sent by customers but have not yet been processed by the retailer.

ground sorts
$\mathbb{N}$ the set of natural numbers

derived sorts
$\mathbf{A} \times \mathbf{B}$ *Cartesian product of sets* ***A*** *and* ***B***
$\boldsymbol{P}(\mathbf{A})$ p*owerset of set* ***A***

function symbols
_+_ : addition on $\mathbb{N}$
_-_ : subtraction on $\mathbb{N}$
_<_ : order on $\mathbb{N}$

multisets
intuitively: a *multiset* is a collection of elements, where an element may occur more than once.
Notation: for example, [a, a, b] or {a, a, b}
Formally: for a given finite set A, a *multiset M over A* is a mapping M: A⟶ $\mathbb{N}$.

operations on multisets
for multisets L and M:
- Sum (union) L+M, with (L+M)(a) = L(a)+M(a)
- For n ∈ $\mathbb{N}$: n•M: n-fold addition of M,
- partial order L < M: component-wise,
$\boldsymbol{P}$(M) power set of of M (set of all sets N ≤ M)
$\boldsymbol{M}$(A) set of all multisets over A

a notation for predicates, p
for a set A: *"p applies to elm(A)"* is shorthand for "*p applies to each element of A*".
For multisets: p applies to all instances.

Fig. 8: Heraklit multisets

We also use a number of variables for the different sorts. Basic and derived sorts, as well as functions and predicates over these sorts, form a *(Tarski) structure*. Such structures form the basis for the formulation of dynamic behavior with Heraklit behavior models.

(Tarski) structures are also the semantic domain for which predicate logic formulates statements. Predicate logic and temporal logic are then close to the formulation and proof of properties of Heraklit models.

Figure 9 depicts the structure and variables we will use to describe the case study below.

IV Distributed runs

Customers submit purchase orders to the retailer and receive deliveries from them; the company reorders any out-of-stock items from a supplier, it assigns a freight forwarder to deliver freight from the supplier to the customer, et cetera. Such behavior, composed of individual events that are considered elementary, is described in Heraklit *behavioral modules*.

9 Local states and events

In the natural sciences and engineering, the behavior of a system is very often modeled as a continuous process along a time axis of real numbers. Here we understand *behavior* fundamentally differently: The behavior of a system is described by *local states*, which are updated by discrete *events*. The result of an event can be the cause for further events.

ground sorts

KN	= {Ute, Max}	*customers*
AR	= {"shoes", "hat", "pants"}	*articles*
WA	= {shoes, hat, pants}	*goods*
TE	= {12/23, 12/24}	*dates*
SP	= {Maier, Müller, Schulz}	*freight forwarders*

derived sorts

AP	= **AR** × ℕ	*items*
AL	= ***M*(AP)**	*article lists*
AM	= ***M*(AR)**	*sets of articles*
WM	= ***M*(WA)**	*sets of goods*

constants

p1: AP	= ("shoes", 2),	
p2: AP	= ("hat", 1)	
K: P(KN)	= {Ute, Max}	*ordering customers*
G: AL	= {("shoes", 3), ("hat", 1)}	*initially listed articles*
H: WM	= {shoes, shoes, shoes, hat}	*initially listed goods*
R: P(SP)	= {Maier, Müller}	*initally available freight forwarders*

functions

f(w) = "w", for w ∈ **WA**

f([a1, ... , an]) = [f(a1), ... , f(an)] for [p1, ... , pn] ∈ **AM**

(a,n)' = n[a] for (a,n) ∈ **AP**

[p1, ... , pn]' = p1' + ... + pn' for [p1, ... , pn] ∈ **AL**

examples

p1' = {"shoes", "shoes"}

[p1, p2, p2]' = {"shoes", "shoes", "hat", "hat"}

Fig. 9: the HERAKLIT structure S of the retailer

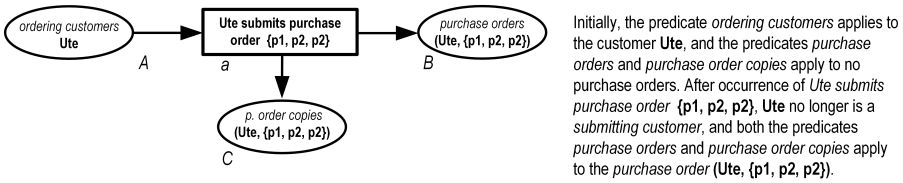


Fig. 10: event *Ute sends order* $\{p1, p2, p2\}$

The description of local states is based on predicates (see section 7): A *local state* is a predicate p together with an object o. Dynamics is created when the object o *reaches* the local state; then p applies to o. If o *leaves* the local state, p does no longer applies to o. Reaching and leaving local states is synchronized by the occurrence of *events*: Some objects reach or leave some local states.

A typical example is the ordering of items: For example, when the customer *Ute* submits an order, she leaves the local state *ordering customers* and the order reaches the two local states *submitted orders* and *copies of submitted orders*. Figure 10 graphically represents the event *Ute submits purchase order* $\{p1, p2, p2\}$: Each of the three ellipses represents one of the involved predicates together with the corresponding object to which the predicate applies. The rectangle t contains the name of the event. An arrow between an ellipse and the rectangle indicates whether the object leaves (arrowhead on the rectangle) or reaches (arrowhead on the ellipse) the corresponding local state when the event occurs.

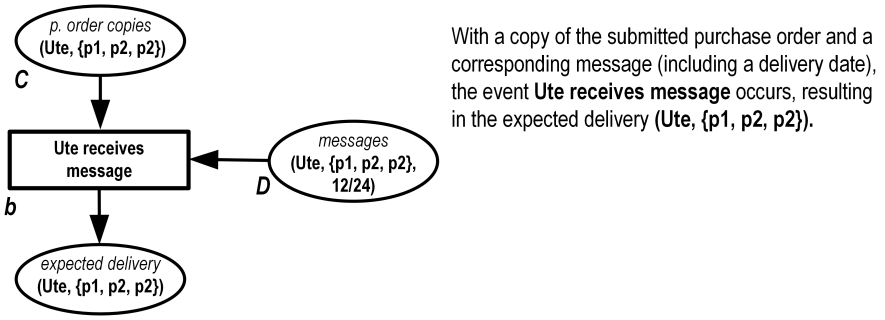


Fig. 11: event *Ute receives message*: delivery arrives on Christmas

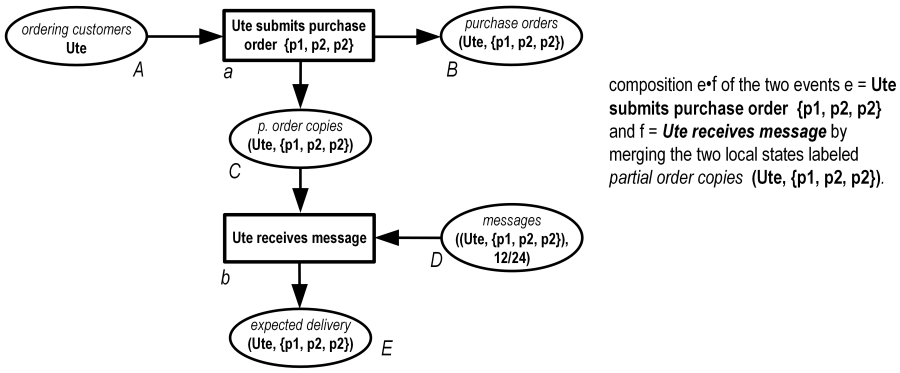


Fig. 12: composition: merging of local states

Formally the event *Ute submits purchase order* $\{p1, p2, p2\}$ has the structure of a *net*: each ellipse is a *place*, the rectangle is a *transition*, and the arrows form the *flow relation*.

Two events e and f can be *composed* to $e \bullet f$, if an object o reaches a local state p by occurrence of e and then leaves it with occurrence of f. As an example, figure 11 shows the event *Ute receives message*. As shown above, the order $(Ute, \{p1, p2, p2\})$ reaches the local state *purchase order copies* (p. order copies) with the event *Ute submits purchase order* $lbracep1, p2, p2\}$ and leaves with the event *Ute receives message*. The two events are composed by merging the two instances of the local state *purchase order copies* as shown in figure 12.

As with a single event, the composition of two events has the structure of a net, consisting of places, transitions and arrows.

10 Distributed runs of modules

The composition of several events describes *complex behavior*; a *run* is created. The composition in figure 12 describes an initial part of a possible *run* of the cus-

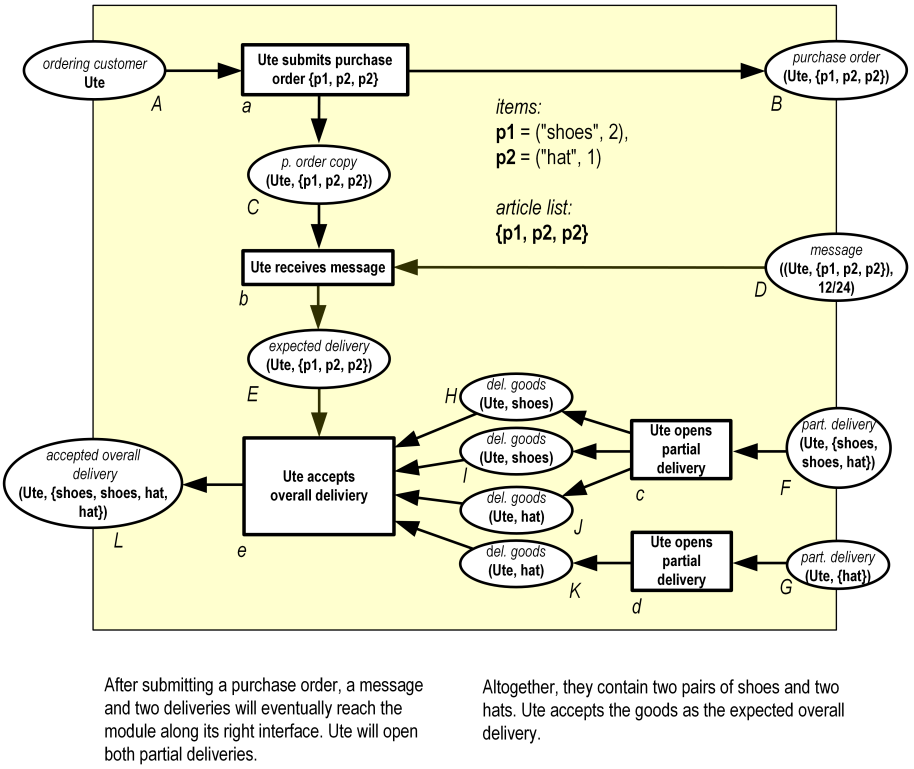


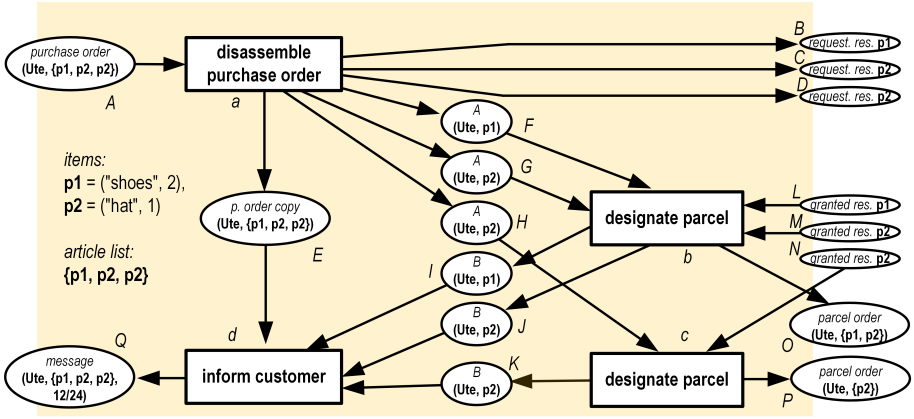
Fig. 13: a run of the module *customers*

tomer module. Figure 13 adds this initial part to a complete run of the customer module. A module can behave in different ways; for example, the customer Ute can choose between very different article lists for her order.

For the run of the module *order management* in figure 14 no order is assumed in which the confirmations of the reservation of the article positions reach the module or in which the module orders the two parcels. However, the customer will be notified only after it has been ensured that the retailer has the appropriate goods in stock for each item position ordered.

The *inventory management* module must answer a request for two pairs of shoes and a request for two hats. In the run shown in figure 14, we assume that the list of available goods of the retailer indicates three pairs of shoes and one hat in stock (this list is briefly, and without technical precision, called a *database*). Thus, a hat must be obtained from the supplier.

The figure 16 shows that the warehouse receives two hats from the supplier and reports this delivery to inventory management. In the run shown, two orders are completed: The first order includes one of the two delivered hats as a parcel and transfers the parcel as freight to the freight forwarders. For the second order,



The order management disassembles the article list of the purchase order into its three items, sends for each item a request for reservation to the inventory management (without naming the client), and retains a copy of the purchase order.

Upon receiving grants for the requested reservations, the order management designates two parcels and send corresponding orders to the warehouse. Finally, the customer **Ute** is informed about the date of the last partial delivery, 12/24.

Fig. 14: a run of the module *order management*

two pairs of shoes and a hat are in stock; they are packed and transferred to the freight forwarders.

As shown in the figure 17, the supplier receives the order for two hats from inventory management and delivers the hats to the warehouse. As figure 18 shows, the two parcels are delivered independently to the customer Ute.

11 Composition of the runs of the modules: a run of the retailer

In section 10 we have seen how individual events can be composed to represent a possible behavior of each of the six modules. Now we compose the behaviors of the modules into a run of the retailer. This run starts with a customer's order, continues through all six modules, and finally returns to the customer with the delivery of the ordered goods. Figure 19 shows the composition of the processes of the six modules from section 10. Two problems arise: First, the graphical arrangement of the interface elements to be merged generally does not match. Second, some interface elements whose labels do not match should be merged. Technically, we organize this with *adapter modules*, graphically represented as a line with a black square, denoted as modules *p1-shoes2* and *p2-hat1* respectively.

As a graph, figure 19 is acyclic: No arrow chain closes to a circle. Some events are thus ordered in a "before-after" relationship: If an event e precedes an event f by a chain of other events, then f is certainly not before e. However, two events may occur side by side with no ordering. This semi-ordering is clearly illustrated by the arrangement of the nodes of the overall run in figure 20: Each arrow runs from left to right, but now parts of individual modules are no longer arranged

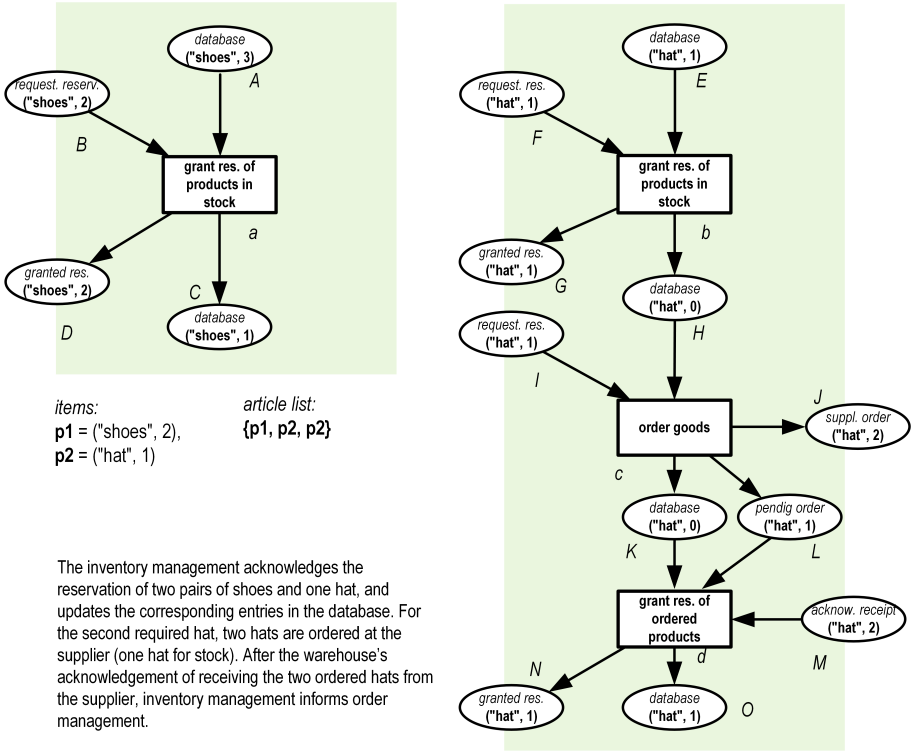
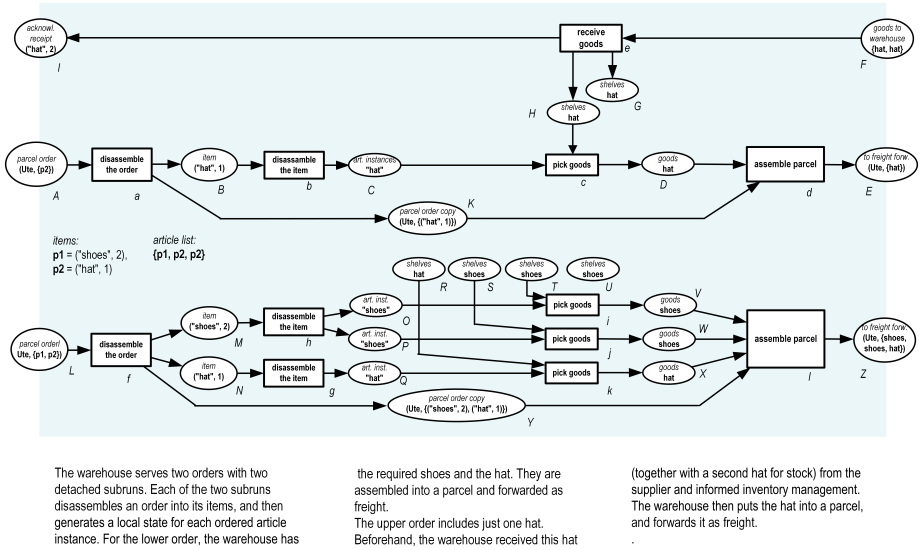


Fig. 15: a run (consisting of two parts) of the module *inventory management*

locally directly together. The colored background shows the contributing module. The flow of some modules is divided into several parts. Occasionally the left/right orientation of interface elements of the modules is swapped.

V Predicates and event schemata

In the fourth part we presented *one* particular run, an example of behavior of a retailer, where the customer *Ute* sends an order with the article list $[p1, p2, p2]$. This is now to be put in more general terms: there are infinitely many possible article lists for an order from Ute; besides Ute, Max can also submit purchase orders; at the beginning, the three articles, namely, hats, shoes, and pants can be available in the warehouse in different quantities, and order management has many different possibilities to put together parcels. Thus, there are infinitely many possible runs. All these runs shall now be formulated in *one* representation. The idea here is to combine events with the same predicates.



The warehouse serves two orders with two detached subruns. Each of the two subruns disassembles an order into its items, and then generates a local state for each ordered article instance. For the lower order, the warehouse has the required shoes and the hat. They are assembled into a parcel and forwarded as freight.
The upper order includes just one hat. Beforehand, the warehouse received this hat (together with a second hat for stock) from the supplier and informed inventory management. The warehouse then puts the hat into a parcel, and forwards it as freight.
.

Fig. 16: a run of the module *warehouse*

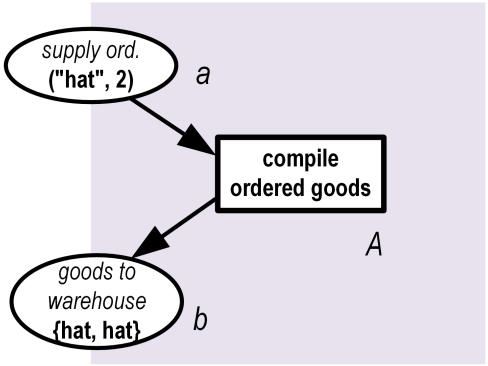


Fig. 17: a run of the module *supplier*

12 Events with equal predicates

First, we represent the event *Ute submits purchase order* $\{p1, p2, p2\}$ from figure 10 differently: as figure 21 shows, the two configurations before and after the event are represented in two different graphs. In figure 21b, the left place does not represent the local state, which consists of the predicate *ordering customers* and the customer *Ute*, but instead represents the predicate *ordering customers*. That this predicate applies to *Ute* is then shown by the token "Ute" within the ellipse. Accordingly, the predicates *sent orders* and *purchase order copies* show that both predicates do not apply to any objects before the event. Figure 21c

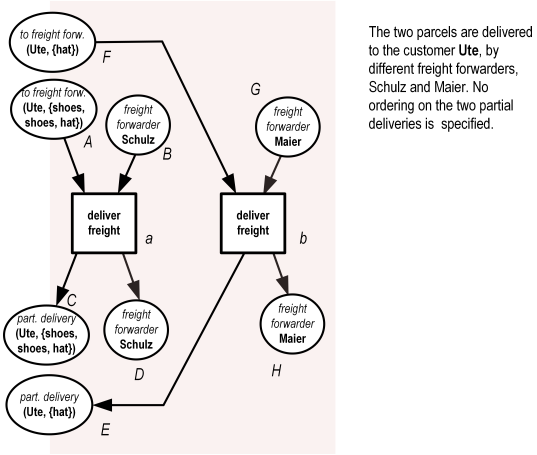


Fig. 18: a run of the module *freight forwarders*

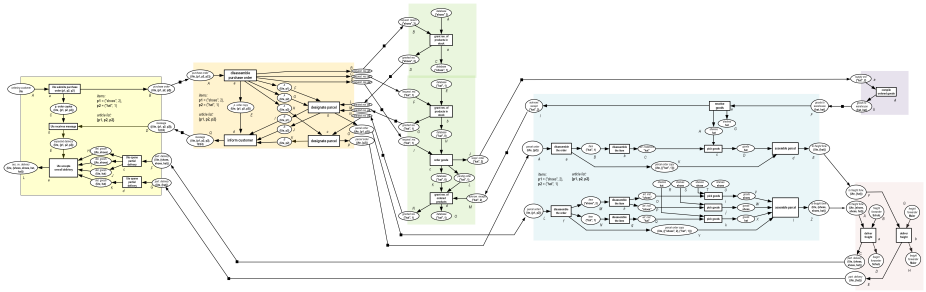


Fig. 19: composition of the runs of the modules: a run of the composed modules (figure is optimized for reading on an electronic device)

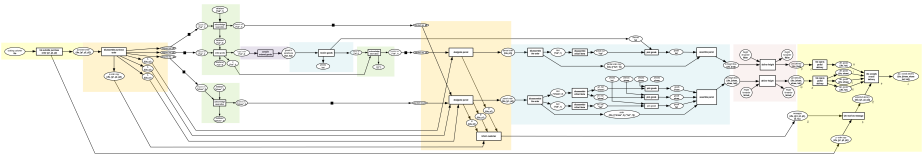
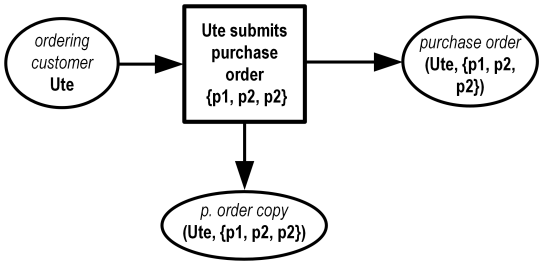
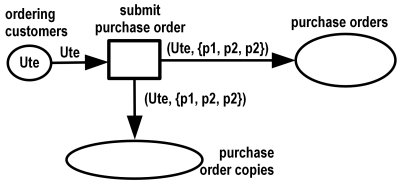


Fig. 20: the run, shown from left to right (figure is optimized for reading on an electronic device)

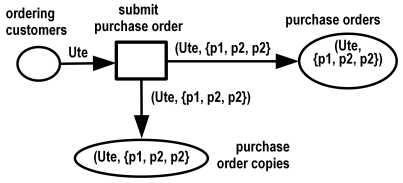
shows the situation after occurrence: the predicate *ordering customers* now does not apply to any object; the predicates *sent orders* and *purchase order copies* now both apply to the object $(Ute, \{p1, p2, p2\})$. From figure 21b alone, one can derive figure 21c: Intuitively formulated, the label on each arrow determines which objects "flow through" the arrow when the event occurs. Thus, figure 21b represents the same behavior as figure 21a.



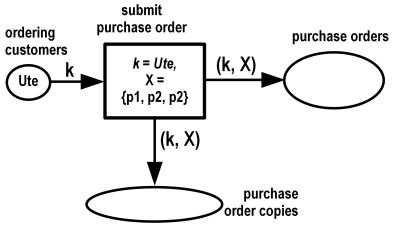
(a) repetition of figure 10: illustration of an event with statements



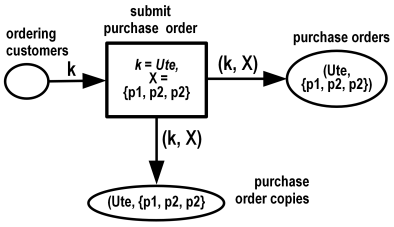
(b) illustration with predicates: before the occurrence of the event



(c) illustration with predicates: after occurrence of the event



(d) illustration with variables: before the occurrence of the event



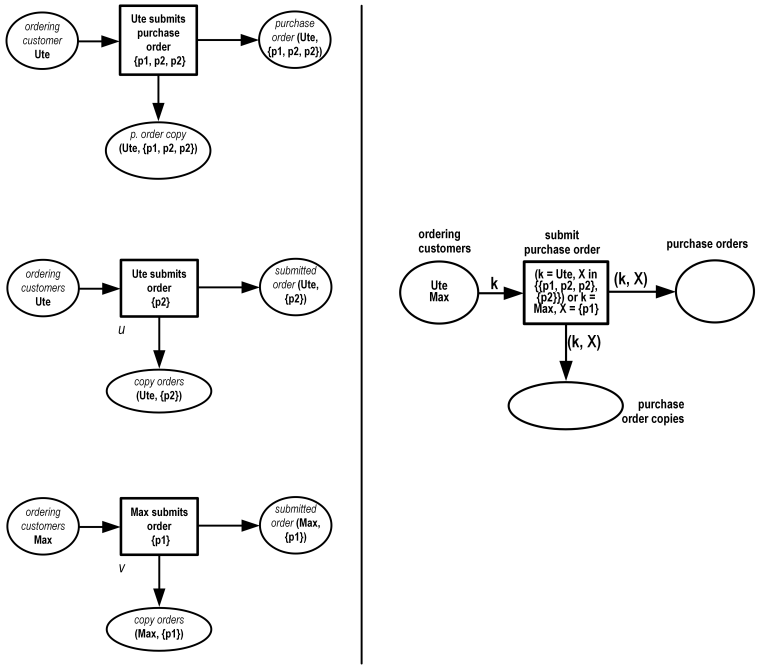
(e) illustration with predicates: after the occurrence of the event

Fig. 21: transition from local states to predicates

In a further step, we replace the arrow labels with variables (figures 21d and 21e), and explain the assignment of the variables in a label of the transition. The representation in figure 21d also represents the behavior of figure 21a.

Figure 22 illustrates the advantage of the new representation. Part 22b shows three different orders: two from Ute and one from Max. Part 22b represents these three orders in *one* schema. The core of the representation is the *variables* k and X. They can be *assigned* with concrete objects: k with a customer (Ute or Max), X with an article list. Three of these assignments fulfill the label of the t transition. Each of them represents one of the three events in figure 22a.

Thus, variables and conditions in the transition t can be used to characterize a set of orders. Particularly simple is the characterization of *all* orders which



(a) three events represented with statements

(b) Common representation of the three events with predicates

Fig. 22: representation of events using predicates

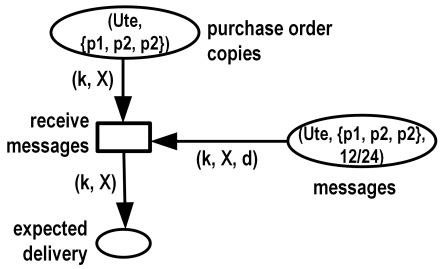
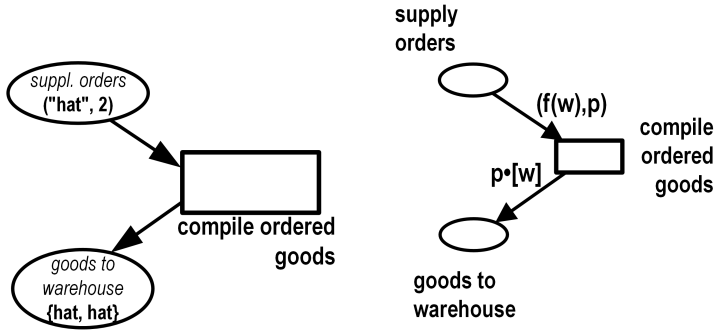


Fig. 23: activated transition *receive messages*

are possible in the given structure: for this purpose any additional condition is omitted.

Figure 23 shows how customers deal with messages from the retailer. The delivery date of the last partial delivery for an order is communicated to the customer: The transition occurs when the variables k and X can be assigned to a customer k_0 and an order list X_0 in such a way that (k_0, X_0) is a token lying



(a) event of the supplier (b) use of the function f

Fig. 24: system functions

on the place *purchase order copies* and for any date d_0 the token (k_0, X_0, d_0) lies on the place *messages*. In figure 23 k, X and d are assigned by tokens $Ute, (p1, p2, p2)$ and 12/24.

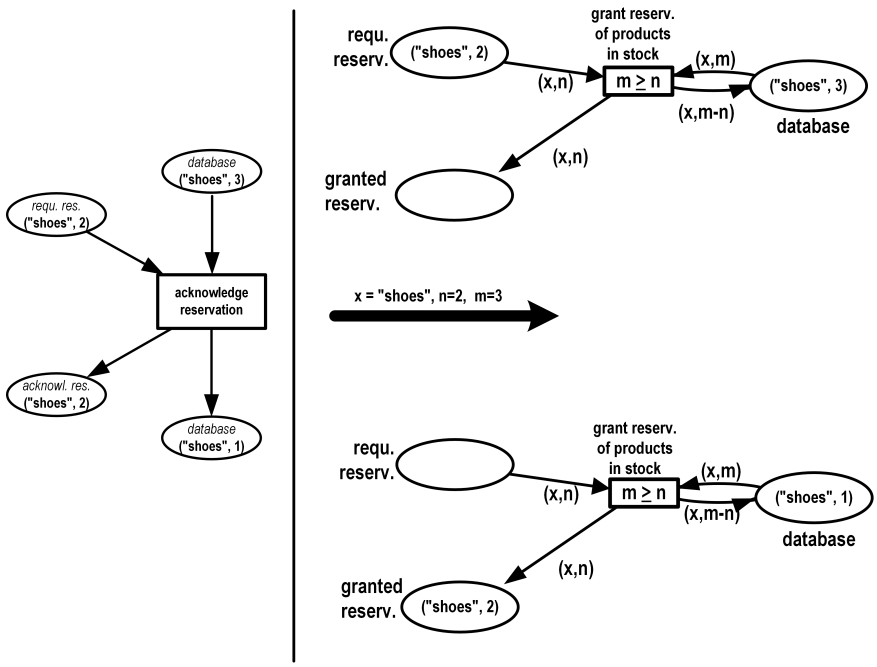
13 Functions

Figure 24a repeats figure 17: The supplier receives the item *("hat",2)* and delivers the multiset $[hat, hat]$. We need a general principle to derive a multiset of goods from an article position. For this we use a function that assigns an article to each good. In our case study this is the function f with $f(hat) =$ *"hat"* and $f(shoes) =$ *"shoes"*. In general, f is not injective; for example, a car rental company could offer small, medium, and large vehicles, three types of goods, but has many real vehicles. Each is small, medium or large.

To schematically characterize the supplier event, figure 24b uses the variable w for the goods to be delivered and the variable p for their number. The single-element multiset $[w]$ with the factor p yields the multiset $p \bullet [w]$, which contains p instances of the goods w. The function $\_'$ returns the corresponding articles for an *item* or *article list*.

14 Loops

It can happen that during an event an object leaves a local state and another (or the same) object reaches a local state with the same predicate. An example of such a predicate is the inventory management database, which is shown again in figure 25a. In a schematic representation, each predicate occurs only once. Therefore, in figure 25b a loop is created between the space *database* and the transition *confirm availability*.



(a) event of the inventory management in the example process

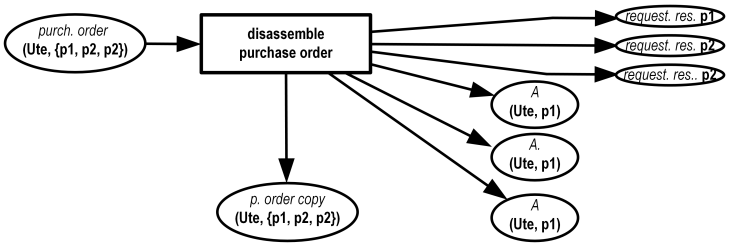
(b) schematic diagram

Fig. 25: event and associated schematic representation with a loop

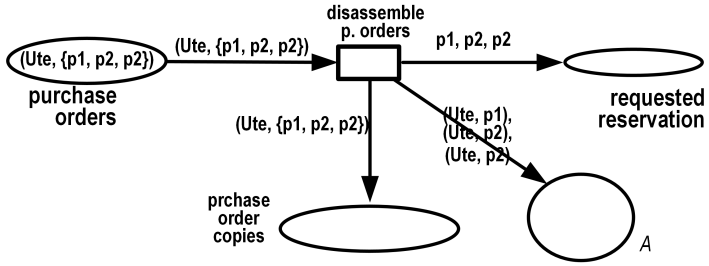
15 elm notation

In the previous examples of schematic representations of events, at most *one* object leaves or reaches a place. This is not always the case. For example, the event *decompose order* in the module *order management* generates three local states with the predicate *requested reservation* (Figure 26a). Therefore, in the schematic representation (figure 26b) the three objects $p1, p2, p2$ are located on the arrow towards the place *requested reservation*. When *disassemble purchase order* (disassemble p. order) occurs, the three objects reach this place at the same time. When using the variable X with the assignment $X = [p1, p2, p2]$, the arrow label X would place the set $[p1, p2, p2]$ as one token on the place instead of the three tokens $p1, p2, p2$, that is, the elements of this set (Figure 26c). The notation $elm(X)$ ensures that after *decompose order* the elements of the set X reach the place *requested reservations*, and not the set itself. The same applies to the place *copy position* and the arrow ending there.

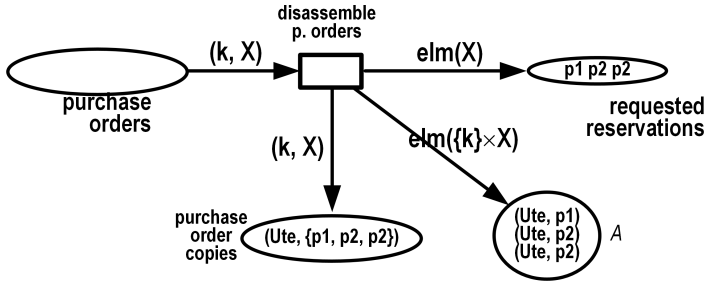
As another example, figure 27 shows how partial deliveries are compared with the order and accepted as a complete delivery: In figure 27a we assume two partial deliveries. First, an incoming delivery is opened and each of the goods



(a) event in module *order management*



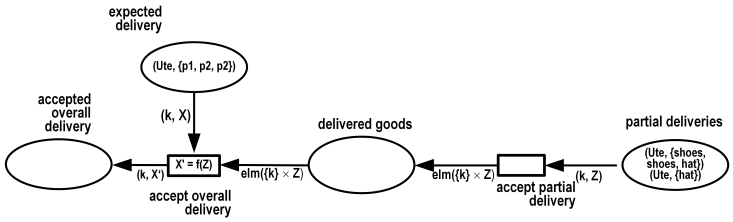
(b) the same event in schematic view



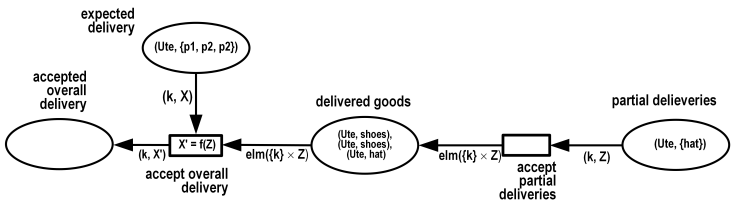
(c) the same event with variables in mode $k = Ute, X = \{p1, p2, p3\}$

Fig. 26: the use of *elm*

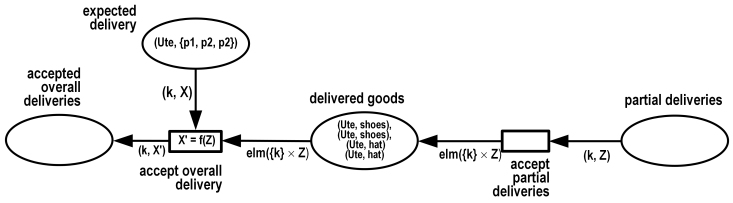
is individually marked with the customer Ute (figure 27b). After the second partial delivery, there are four goods on the place *delivered goods* (figure 27c). The quantity of these goods is assigned to the variable Z; $f(Z)$ indicates the quantity of the ordered articles. This quantity is derived with X' from the article list $X = [p1, p2, p2]$. With this, the condition in the transition is fulfilled, the transition occurs and the complete delivery is accepted (figure 27d).



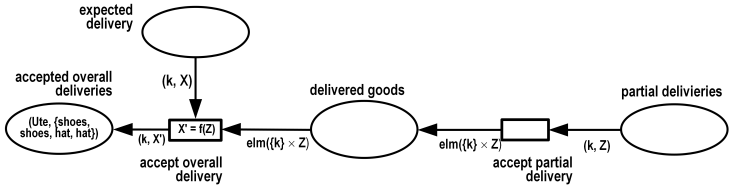
(a)



(b)



(c)

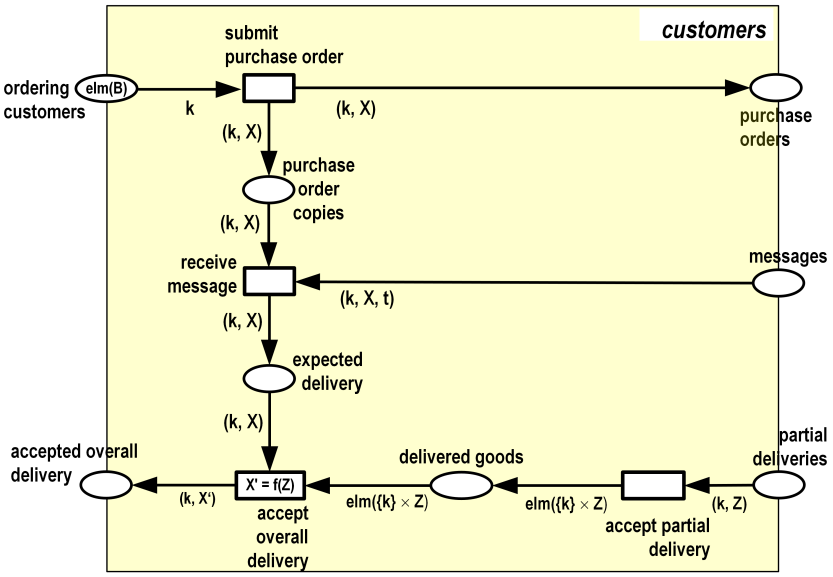


(d)

Fig. 27: receive goods

VI The retailer for the structure S

We now have all the means of expression at our disposal to model all aspects of the case study. In the logic of an order and its processing, we start with the customer model. Then we follow with the three departments of the retailer (order management, inventory management and warehouse) and finally the two other business network partners: the supplier and the freight forwarders. The models are based on structure S, which was presented in section 8, figure 9.



Place **ordering customers**: This place initially contains a token for each potential customer. A purchase order consists of a customer k, and an article list X.

Transition **receive message:** This transition ocurs if the received message is identical with a copy of a previously submitted order, extended by a date.

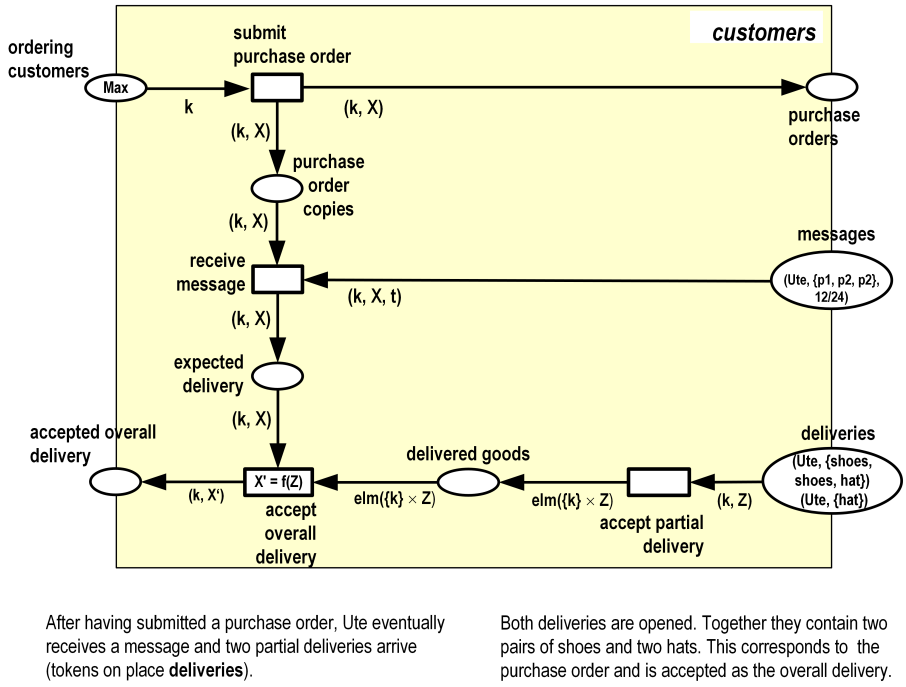
Transition **accept partial delivery:** A partial delivery Z for the customer k is opened, and each single good in Z turns into a token of its own. There may arrive many such partial deliveries for the same customer, k.

Transition **accept overall delivery:** This transition occurs when, for each item on the article list in the place **expected delivery,** there is a corresponding number of goods on place **delivered goods**.

Fig. 28: the module *customers*

16 The customer model

Figure 28 shows the module *customers*. With the set $K = \{Ute, Max\}$ from the structure S, the two tokens Ute and Max are located at the beginning in the place *ordering customers*. We have already discussed the transitions in detail. In the context of the other modules, figure 29 shows a reachable marking from which Ute receives a message with the delivery date (12/24) and then receives and accepts the entire delivery in two partial deliveries, as discussed in figure 27. Independent of this, Max can submit a purchase order.



After having submitted a purchase order, Ute eventually receives a message and two partial deliveries arrive (tokens on place **deliveries**).

Both deliveries are opened. Together they contain two pairs of shoes and two hats. This corresponds to the purchase order and is accepted as the overall delivery.

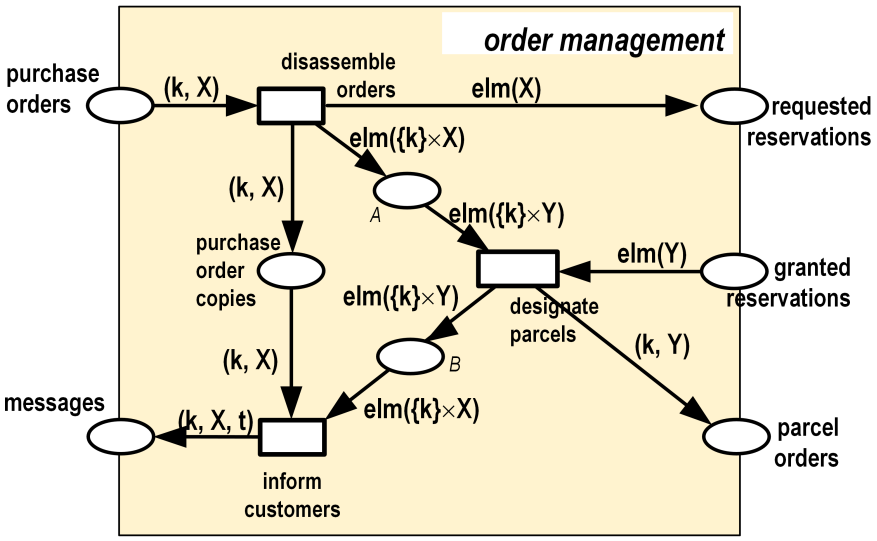
Fig. 29: a reachable marking of the *customer* module in the context of all modules

17 The order management model

We have already discussed the transition *decompose order* of the order management (Figure 30) in figure 26. The order management asks for a confirmation of the availability of each individual order item. This can be delayed if, for example, some units have to be reordered from the supplier first. As soon as some order items have reached the place *confirmation of availability*, the order management assigns the warehouse a corresponding partial delivery. The order items sent in partial deliveries are collected on place C; when all items of an order are assigned to partial deliveries, the transition *notify customer* sends a message to the customer with the scheduled date for the last partial delivery of the order. Technically, the order management can assign any date to the variable t.

18 The inventory management model

In the inventory management model (figure 31), the entries of the article list G in place A describe, for each ordered article, the number of matching available goods in the warehouse, initially three pairs of shoes, a hat and no shirts. If there are at least n units available in the warehouse for a requested order item (a, n), transition a releases n units for a partial delivery. If not, transition b orders the



Transition **disassemble orders**: The order management disassembles the article list X of an incoming order (k,X) of a customer k into its items; each of these items is placed on place **requested reservations**, as well as on place *A*. Each item in *A* is additionally indexed with the customer k.

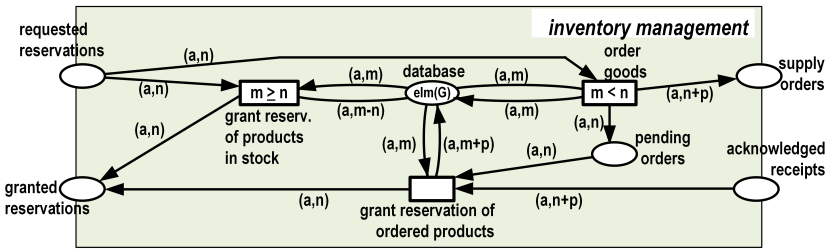
Transition **inform customers**: Place *B* collects the items of partial deliveries fo customer k . When the items complete the purchase order, the customer is informed about the date t of the last partial delivery.

Of particular interest is the transition **designate parcels:** Order management selects a subset Y of items from all granted reservations. Then a parcel for Y is designated, and all items in Y are collected in place *B*.

Fig. 30: order management queries availability of goods and orders partial deliveries

n units of the article a from the supplier as well as a stock of a further p units. By defining p as a variable, inventory management can freely decide the amount of stock for item a for each reorder. If p were declared as a constant, the S structure would set the stock level for the item a indefinitely.

When the request to restock an item triggers an order to the supplier, the request is kept on place B until the warehouse has confirmed receipt of the units from the supplier. The transition c then passes this confirmation to the order management and updates the information about the stock in place A. Place A thus describes for each item how many units have not yet been reserved for



Place **database** reports the goods in the warehouse. This list is a set **G** of items, indicating for each good the corresponding number of available instances. Each such item is a token on the place **database.**

A request for the reservation of an item **(a,n),** i.e. for n instances of an article **a**, has two potential outcomes:

(1) The number m of available instances meets or exceeds the number n of requested instances. In this case, the transition **grant reservation of produts in stock** grants reservation of the goods, and reduces the corresponding number in the corresponding item of **database.**

(2) Otherwise, inventory management orders n+p instances from the supplier, increasing the number n of requested instances by p, as stock for forthcoming inquiries. The inventory management may select a differrent number p at each occurrence of **order goods.**

The warehouse will eventually acknowledge receipt of n+p instances. Then, the inventory management updates the corresponding item in **database**, and acknowledges reservtion of n instances to order management.

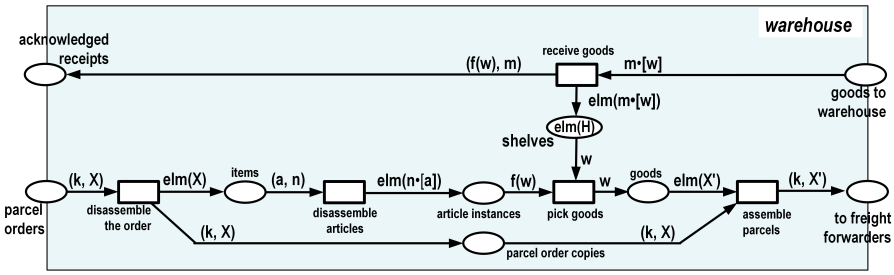
Fig. 31: inventory management confirms the availability of article

delivery in the warehouse. Here, one can see a subtle difference between one large order quantity and several small order quantities of a particular item: If, for example, three shirts are available and one order requests four shirts, that whole request is held back on place B until the supplier has delivered more shirts. If, instead, two orders requesting two shirts each are made, one of them is confirmed immediately and in a partial delivery the customer can already receive the first two shirts before the supplier delivers the other shirts to the warehouse.

19 The model of the warehouse

The warehouse in figure 32 has two tasks: for each incoming order it packs a freight parcels for the freight forwarders; furthermore, the warehouse processes the incoming deliveries from the supplier. For the first task, the transition a breaks down an incoming article list into its items. From the item (a, n) of an article a, which is ordered in quantity n, the transition b then creates n tokens of the type a: first, $n \bullet [a]$ creates the multiset containing n instances of the type a. With $elm(n \bullet [a])$, each of its n elements then becomes an individual token. For each of these tokens a, the transition c takes a good w from the warehouse C, which corresponds to a, to which it therefore applies that $f(w) = a$. This commodity is put on place D.

For the second task the transition d receives a parcel $m \bullet [w]$ from the supplier with m instances of the good w. The instances are individually stored in the warehouse ($elm(m \bullet [w])$ is used to create m tokens of the type w). At the same



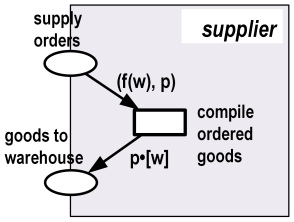
This module performs two tasks: 1. According to the orders coming from *order management*, the module assembles parcels and forwards them to the *freight forwarders*. 2. The module receives incoming goods from the *supplier* and informs *inventory management*.

For the first task, the module stores each incoming order in place **parcel order copies**, and additionally disassembles the order into its **articles**. Then, transition **disassemble article items** generates, for each item (a,n) of an article, n tokens of the form "**a**" (instances of a) on the place **article instances**.

For each item, transition **pick goods** picks corresponding goods from the shelves and forwards them to place **goods**. The function f describes the goods w that would correspond to the article f(w). Transition **assemble parcels** bundles the goods according to the article list **X** and forwards the parcel to the *freight forwarders*.

For the second task, the transition **receive goods** loads the instances of an incoming goods delivery into the **shelves**, and informs *inventory management*.

Fig. 32: warehouse packs freight parcels and confirms the goods sent by the supplier



Transition **compile ordered goods**: From the inventory management, the supplier receives the order of p instances of article a, and delivers p instances of "matching" goods w. A good w "matches" an article a if f(w) = a.

Different goods may match the same article: the goods may be shaped differently or may have different contol numbers, etc.

Fig. 33: supplier sends corresponding goods for each order

time, inventory management is informed about the receipt of the goods: $f(w)$ describes the article of the goods w, $(f(w), m)$ also the quantity.

Place C represents the actual stock; it initially contains the elements of the multiset H, that is three pairs of shoes, a hat and no shirts. This corresponds to the initial marking $elm(G)$ of place A of the inventory management.

20 The supplier model

Figure 33 shows how the supplier receives a supplier order and selects an item w that matches the ordered item $f(w)$. The supplier then transports a package

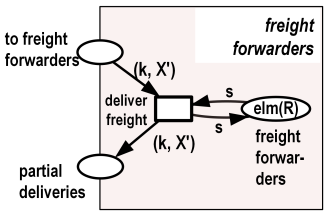


Fig. 34: freight forwarder delivers freight parcels

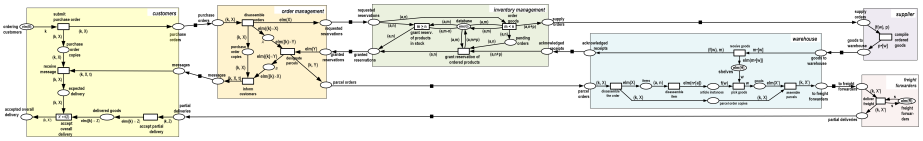


Fig. 35: model of the retailer, composed of six modules (figure is optimized for reading on an electronic device)

of p instances of this item to the goods receiving area of the warehouse (token $p \bullet [w]$). In this model, the supplier can always deliver; of course, it is possible to model other assumptions here.

21 The model of the freight forwarders

The model of the freight forwarders in figure 34 is obvious: There is a basic type R of freight forwarders. Initially, the place A contains some freight forwarders. The variable s is used to select a freight forwarder. This module provides a starting point for further assumptions regarding the selection of freight forwarders.

All modules of the case study are now modeled. The models can be composed as shown in figure 35. This creates the model of the retailer for the structure S.

VII The case study model

In the model shown in figure 35 the basic sorts and basic functions, which are the customers, goods, initially available goods, assignment of articles to goods, and the freight forwarders are fixed. A model that includes all possible basic sorts and basic functions would be interesting. Furthermore, the model itself should be purely *symbolic*, so it should not work with specific quantities and functions like the models in section VI, but with symbols for quantities and functions. Then the model can be processed with software tools.

Here we can again take up proven concepts of predicate logic and many specification languages: We construct a *signature* Σ for the retailer. The basic idea is simple: Σ contains a symbol for each basic sort, each derived sort, each constant and each function. An *instantiation* of the signature assigns a matching

ground sorts
KN *customers*
AR *articles*
WA *goods*
TE *dates*
SP *freight forwarders*

derived sorts
AP = **AR** $\times$ $\mathbb{N}$ *items*
AL = ***M***(**AP**) *article lists*
AM = ***M***(**AR**) *sets of articles*
WM = ***M***(**WA**) *sets of goods*

constant symbols
p1, p2: AP *ordered article positions*
B: P(KN) *ordering customers*
G : AL *initially listed articles*
H : WM *initially available goods*
R : P(SP) *available freight forwarders*

function symbols
f : WA $\longrightarrow$ **AR**
f : WM $\longrightarrow$ **AM**
(_'): AP $\longrightarrow$ **AM**
(_'): AL $\longrightarrow$ **AM**

variables
k: KN
x: AR
X,Y: AL
Z: WM
t: TE
w: WA
s: SP
m, n, p: $\mathbb{N}$

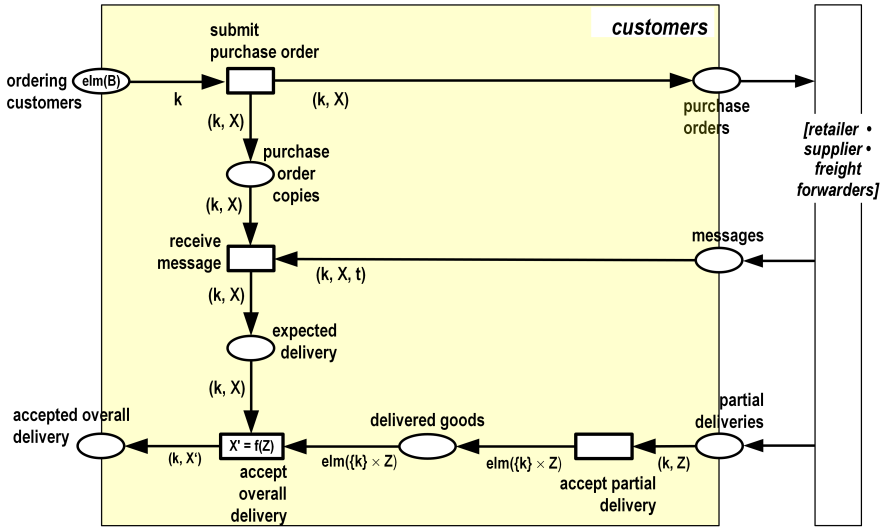
properties
(a,n)' = n[a] for (a,n)$\in$ **AP**
[p1, ... , pn]' = p1' + ... + pn' for
[p1, ... , pn] $\in$ **AL**

G' = **f(H)**

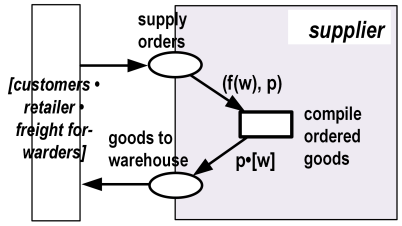
Fig. 36: Signature Σ

set or a function to each such symbol. Figure 36 shows such a signature. An *instantiation* then assigns a set, object or function to each symbol according to its type. The structure S is an instantiation of the signature Σ.

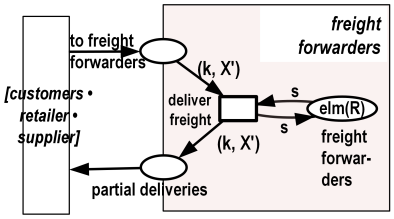
To describe the behavior of instantiations, we can use the nets from part VI as they are. This is because these nets do not use specific aspects of the instantiation S; for example, the initial marking of the place *ordering customer* is not labeled with Ute and Max, but $elm(K)$. The instantiation S then results in Ute and Max as the initial marking of the place *ordering customers*. Somewhat more difficult is the general formulation of the relationship between stored positions of available items in inventory management and the available goods in the warehouse: The function f, which can be freely chosen, describes which different goods fulfill an ordered item request.



(a) perspective of the customer



(b) perspective of the supplier



(c) perspective of the freight forwarder

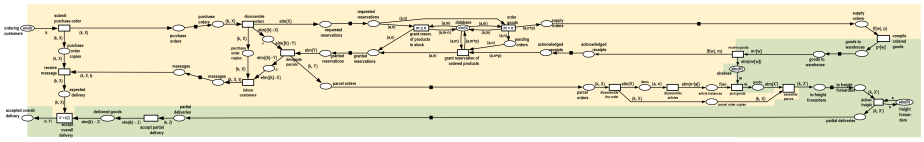
Fig. 37: perspectives of the business network partners on their respective environment

The concepts and notations are chosen according to the specification language Z. Z offers concepts for composition, import et cetera of specifications. The very rudimentary concepts for describing dynamic behavior in Z are replaced by HERAKLIT with its sophisticated behavior models. While predicate logic deals with properties of structures, HERAKLIT describes the dynamic modification of structures, particularly the extension of predicates. Thus, HERAKLIT is the natural dynamic extension of static predicate logic.

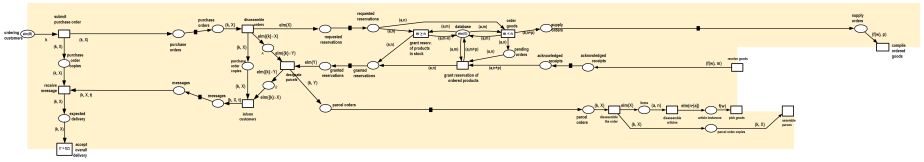
VIII Further perspectives on the retailer

22 Abstract environments for behavioral models

There are a number of other perspectives on the retailer and its business network partners. For the behavioral module of each partner there is an abstract module,



(a) flow of orders and goods



(b) module *orders*

(c) module *flow of goods*

Fig. 38: modules of orders and flow of goods (figure is optimized for reading on an electronic device)

its environment, so that the composition of both modules results in the overall module. Figure 37 shows the behavior of the retailer's three business network partners in their respective environments.

23 Module for orders and flow of goods

Figure 35 composes the six modules involved in the retailer and thus describes the behavior of the entire system in a structured way: The components remain recognizable. However, the overall module can also be structured differently. Figure 38a separates that part of the model which represents real physical objects, namely goods, from the part that organizes abstract data of the orders. This separation leads to the two modules in figure 38b and 38c. Their composition again results in the overall module in figure 38a. It therefore holds that:

$$overall\,behavior = orders \bullet flow\ of\ goods.$$

Appendix

24 Theoretical, conceptual and methodological foundations

The central concept of a HERAKLIT module combines three proven, intuitively easy to understand and mathematically well-founded techniques, which are already used for the specification of systems:

1. *abstract data types* and *algebraic specifications for the formulation of concrete and abstract data*: Since the 1970s, such specifications have been used, built into specification languages, and often employed for (domain-specific) modeling. These include the older languages *VDM* and *Z* [9], [2], the rich language *RAISE* [6], [1] and many other languages. [10] tries to standardize the core of the field with the Common Algebraic Specification Language CASL. Very systematically, the book [15] presents the theoretical foundations and some fields of application of algebraic specifications. *Abstract State Machines* (ASM) [7] belongs to the same context: Starting with a signature Σ there are global states; each state is an instantiation of Σ. A step is typically a conditional value assignment to a variable or term whose value is updated this way. Similarly, HERAKLIT describes states, but *local* state components, with instantiated basic types. For a type A or a function f of an instantiation, a local state component is a subset or a single element of A, or a tuple of the form $(a, f(a))$.
2. *Petri nets for the formulation of dynamic behavior*: HERAKLIT uses the central ideas of Petri nets: A step of a system, especially a large system, has *local limited* causes and effects. This allows one to describe processes without having to use global states and globally effective steps. This concept from the early 1960s [11] was generalized in the early 1980s with *predicate logic* and *colored tokens* [5], [8]. The connection with algebraic specifications is established by [12]. HERAKLIT adds two crucial aspects to this view: uninterpreted constant symbols for sets in places that use the *elm* notation to cover instantiations with many possible initial tokens, and the elm notation in arrow labels to describe flexible token flow.
3. *The Composition Calculus for Structuring Large Systems*: this calculus, with its widely applicable associative composition operator, is the latest contribution to the foundations of HERAKLIT. The obvious idea, often discussed in the literature, to model composition as a fusion of the interface elements of modules is supplemented by the distinction of left and right interface elements. The composition $A \bullet B$ is defined as a fusion of the right interface elements of A with the left interface elements of B. According to [13], this composition is associative (in contrast to the naïve fusion of interface elements); it also has a number of other useful properties. In particular, this composition is compatible with refinement and abstraction and with distributed runs. Further details and examples are discussed in [14] and [3].

These three theoretical foundations harmonize with each other and generate further *best practice* concepts, which contribute to a methodical approach to modeling with HERAKLIT, and which are only touched upon in this paper. These include in particular the notion of an *adapter*: An adapter for two modules A and B formulates criteria and properties for the composition of A and B as another module C, so that $A \bullet C \bullet B$ guarantees the desired properties. Further details are described in [14] and in the HERAKLIT manual. The methodical, systematic use of the composition of abstraction and of distributed runs will be demonstrated in future case studies.

References

1. D. Bjørner. *Domain Science & Engineering: A Foundation for Software Development*. Unpublished manuscript (2020).
2. J. P. Bowen. Z: A formal specification notation. In *Software specification methods*, pages 3–19. Springer, 2001.
3. P. Fettke and W. Resig. Modelling service-oriented systems and cloud services with HERAKLIT. *CoRR*, abs/2009.14040, 2020. presented at the 16th International Workshop on Engineering Service-Oriented Applications and Cloud Services, Heraklion, Greece, September 28-30, 2020.
4. P. Fettke and W. Resig. Modellieren mit HERAKLIT – handbuch (in vorbereitung). 2021.
5. H. J. Genrich and K. Lautenbach. The analysis of distributed systems by means of predicate/transition-nets. In G. Kahn, editor, *Semantics of Concurrent Computation, Proceedings of the International Symposium, Evian, France, July 2-4, 1979*, volume 70, pages 123–147. Springer, 1979.
6. C. George. The NDB database specified in the RAISE specification language. *Formal Asp. Comput.*, 4(1):48–75, 1992.
7. Y. Gurevich. Evolving algebras 1993: Lipari guide. In E. Börger, editor, *Specification and validation methods*, pages 9–36. Oxford University Press, 1993.
8. K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use - Volume 1*. Springer, 1992.
9. C. B. Jones. *Systematic software development using VDM*. Prentice Hall, 2 edition, 1991.
10. T. Mossakowski, A. E. Haxthausen, D. Sannella, and A. Tarlecki. Casl - the common algebraic specification language: Semantics and proof theory. *Comput. Artif. Intell.*, 22(3-4):285–321, 2003.
11. C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Technische Hochschule Darmstadt, 1962.
12. W. Reisig. Petri nets and algebraic specifications. *Theor. Comput. Sci.*, 80(1):1–34, 1991.
13. W. Reisig. Associative composition of components with double-sided interfaces. *Acta Informatica*, 56(3):229–253, 2019.
14. W. Reisig. Composition of component models - a key to construct big systems. 2020. in press.
15. D. Sannella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Springer, 2012.