Case Study Logistics: Flexible Modeling of Business Processes using Algebraic Higher-Order Nets

Kathrin Hoffmann

Institute for Software Engineering and Theoretical Computer Science
Technical University Berlin, Germany
email : hoffmann@cs.tu-berlin.de

Abstract. In this paper we present a case study logistics using Algebraic Higher-Order Nets. Algebraic Higher-Order Nets are an extension of the well defined formalism of Algebraic High-Level Nets by higher-order functions leading to a more flexible modeling technique. After an introduction of Algebraic Higher-Order Nets at an informal level we present the case study logistics and demonstrate the advantage of our approach which allows a flexible modeling of business processes including exceptions and roles without changing the net structure of our nets.

1 Introduction

In the context of parallel and concurrent systems Petri nets represent a well-known and widely used formalism and have been employed in practical applications in different areas. Their graphical representation and formal semantics excellently support the modeling and simulation of such systems. Among a large variety of different high level net classes (see e.g. [Gen91,Jen92]) Algebraic High-Level Nets [EPR94] give rise to a formal and well defined description of the dynamic behaviour of concurrent and distributed systems due to their combination of algebraic specification and Petri nets. This formalism is adequate in application domains, where the context is known from the very beginning such that the system can be modeled by a Petri net with a fixed structure. In other application domains like business process modeling it is also desirable to support the fact that an organization of a system is not fixed once and for all, that is, changes of the environment can only be modeled by changing the structure of Petri nets.

In the area of Petri nets higher-order structures as tokens is a topic of research (see e.g. [Han97,Val98,Hof00]) leading to a more flexible modeling of business processes. In this paper we show that our concept of Algebraic Higher-Order Nets [Hof00] including algebraic higher-order signatures as data type part [MHKB99] is suitable for flexible modeling of business processes.

The paper is organized in the following way: In Section 2 we illustrate the main idea of Algebraic Higher-Order Nets with a small example using a simplified notation of algebraic higher-order signatures, specifications and algebras.

H. Ehrig et al. (Eds.): Petri Net Technology ... , LNCS 2472, pp. 145-160, 2003.

Subsequently we present our case study logistics using Algebraic Higher-Order Nets (Section 3), which allows a flexible modeling of business processes without changing the net structure (Section 4), and a short conclusion (Section 5).

2 Motivation and Informal Introduction

In this section we motivate the notions of Algebraic High-Level Nets and Algebraic Higher-Order Nets in terms of a small example in order to demonstrate in the following section how Algebraic Higher-Order Nets are used for flexible modeling of business processes without changing the net structure.

In the application domain of business processes Petri nets are a widely used formalism and have been successfully employed in practical applications. For the integration of data elements into business processes it is adequate to use at least High-Level Petri Nets. In this way we obtain a more compact model, because a particular part of the net structure is reduced to data elements.

Let us first consider Algebraic High-Level Nets [PER95], an integration of classical Petri nets and algebraic specifications [EM85]. Technically, the algebraic specification is used to define net inscriptions by terms over the specification. Furthermore firing conditions defined by algebraic equations guarantee, that certain constraints are respected. The marking of Algebraic High-Level Nets consists not only of black tokens, but also of data tokens which are elements from a given algebra. In contrast to black tokens of low level Petri nets data tokens can be modified during the firing of transitions.

Figure 1 shows a very simple Algebraic High-Level Net. The idea of the net is to compute alternatively the successor, the square or the cube of natural numbers. The net is inscribed with terms over the signature $\underline{\underline{NAT}}$ of natural numbers given below.

$$
\begin{array}{ll}
\underline{\underline{NAT}} = & \\
\underline{sorts}: & nat \\
\underline{opns}: & zero :\rightarrow nat \\
 & succ : nat \rightarrow nat \\
 & square : nat \rightarrow nat \\
 & cube : nat \rightarrow nat
\end{array}
$$

We consider the $\underline{\underline{NAT}}$-algebra A consisting of natural numbers with constant $zero_A = 0$ and the well-known functions $succ_A, square_A$ and $cube_A$, respectively. In Figure 1 the type of place `p1` is given by the sort nat and the initial marking of place $p1$ consists of natural numbers $n_1, \ldots, n_k \in \mathbb{N}$.

To demonstrate the firing behaviour of the transition `Compute succ` (and similar for `Compute square` and `Compute cube`) we first assign a natural number n to the variable x. The follower marking is computed as follows: the data element n is consumed from place `p1` and the result $succ_A(n)$ is added to the place `p2`.

The Algebraic High-Level Net `Computation` in Figure 1 can be represented in a more compact way by an Algebraic Higher-Order Net `Computation'` in

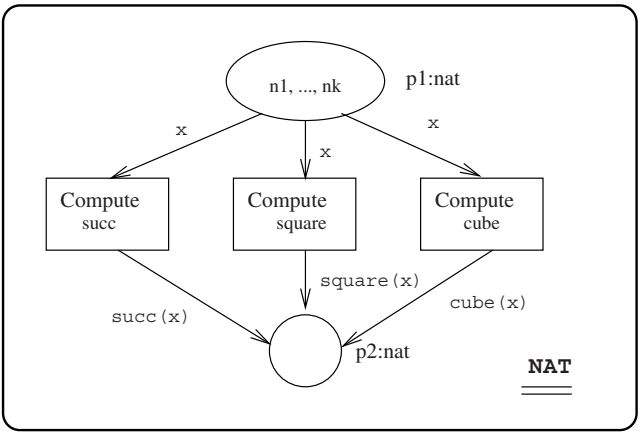

Fig.1: Algebraic High-Level Net `Computation`

Figure 2, where the three different transition in Figure 1 are presented by one transition together with a higher-order place `p` and three different function type tokens $succ, square$ and $cube$. The function type in this case is $nat \to nat$ which means that $succ, square$ and $cube$ can be interpreted as functions from natural numbers to natural numbers. In order to fire transition `Computate` in Figure 2 we have to give not only an assignment to variable x of type nat (as in Fig. 1), but also an assignment to variable f of type $nat \to nat$. Assigning n to x and $succ$ to f we can fire transition `Computate`, where token n is removed from place p_1, token $succ(n)$ is added to place p_2, but token $succ$ remain on place p as indicated by the horizontal line between transition `Computate` and place p. In fact, place p can be considered as a context place in the sence of contextual nets (see [MR95]), where, however, p is now a place of higher-order type.

In order to allow places and tokens of function type we have to replace the algebraic signature NAT by an algebraic higher-order signature HO-NAT and the NAT-algebra A by a HO-NAT-algebra $HO\text{-}A$. This is possible in the framework of Algebraic Higher-Order Nets, which is formally introduced in our paper [Hof00] bases on algebraic higher-order specification (see [MHKB99]). For the presentation in this paper, however, we use a most simplified notation for higher-order signatures and algebras as follows:

$$
\begin{array}{ll}
\underline{\underline{HO-NAT}} = & \\
\underline{basic\ \ sorts}: & nat \\
\underline{opns}: & zero :\to nat \\
 & succ : nat \to nat \\
 & square : nat \to nat \\
 & cube : nat \to nat
\end{array}
$$

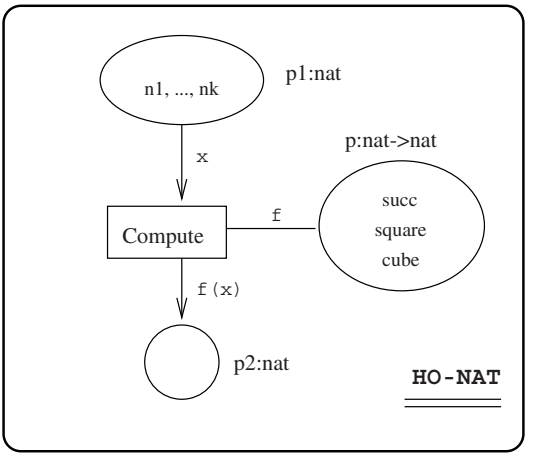

Fig.2: Algebraic Higher-Order Net `Computation'`

where $succ, square$ and $cube$ - written as operation symbols from nat to nat - are in fact higher-order function symbols of type $nat \to nat$. This notation of operation symbols allows to build up terms like $succ(x), square(x)$ and $cube(x)$ with the usual interpretation, or more general $f(x)$, where f is a variable of type $nat \to nat$. In addition to the basic sort nat HO-NAT has also product and function types like $nat \times nat, nat \to nat$ and $(nat \to nat) \times nat$, where in Fig. 2 we only need the basic sort nat and the function type $nat \to nat$. This allows to consider $succ, square$ and $cube$ as higher-order constants of type $nat \to nat$, which are used as tokens in place p in Fig. 2.

For a precise notation of higher-order signatures, algebras and Algebraic Higher-Order Nets we refer to [MHKB99,Hof00]. In [Hof00] we have also introduced two types of transformations between specific Algebraic High-Level Nets and Algebraic Higher-Order Nets, called folding and unfolding. In our case the Algebraic Higher-Order Net in Fig. 2 is the folding of the Algebraic High-Level Net in Fig. 1, and vice versa the Algebraic High-Level Net in Fig. 1 is the unfolding of the Algebraic Higher-Order Nets in Fig. 2.

The advantage of Algebraic Higher-Order Nets is the reusability of the fixed net structure. To include further computations of function type $nat \to nat$ into the Algebraic Higher-Order Net `Computation'`, we extend the marking of place `p`, but can keep the same net structure. In the Algebraic High-Level Net `Computation` the net structure would have to be extended by one transitions for each further computation.

In Figure 2 we have only used the basic sort `nat` and the function type $nat \to nat$. As explained above we can also use product types and combined function and product types.

In Figure 3 the type of place `p` is given by the product type $(nat \to nat) \times person$. The functions as given by the marking of place `p` are extended by roles

`r1` and `r2`, respectively. This means, that e.g. the functions *succ* can only be applied by a person in role `r1`, while the firing behaviour is equivalent to the Algebraic Higher-Order Net `Computation'`.

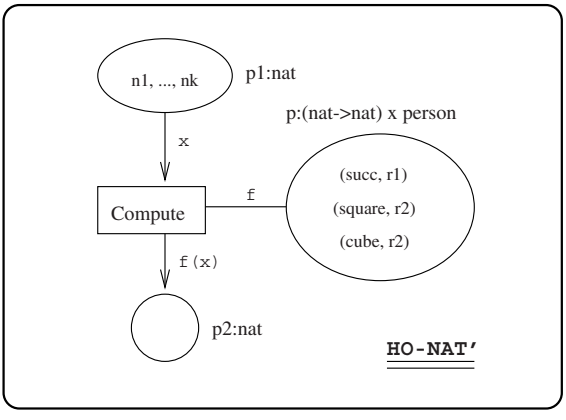

Fig: 3: Place with Product Type

Summarizing Algebraic Higher-Order Nets give rise to a more compact model by using functions as tokens. The advantage is the flexible modeling of business processes, that means that changes of the environment are modeled by changing the corresponding marking. Furthermore we are able to introduce a role model into Algebraic Higher-Order Nets.

3 Case Study Logistics

The business process logistics consists of the planning and scheduling functions concerned with the distribution of products to customers. In view of departments the business process can be divided into different parts, each of them with specific documents and activities. After receiving an order from a customer in the order department the availability of articles is checked. In the delivery department the corresponding delivery note is generated and articles are send to the customer. Finally in the accounts department invoices are captured and cleared with incoming payments.

Our case study logistics is based on data structures defined in [Sch94] by entity/relationship-diagrams and processes modeled by event driven process chains. In more detail our case study logistics consists of three Algebraic Higher-Order Nets `Order Department`, `Delivery department` and `Account department` corresponding to the departments described above. To obtain the overall system of the business process logistics we first observe, that there are places with the same name which are present in different Algebraic Higher-Order Nets. Our Algebraic Higher-Order Nets are sequentially composed by

merging places `checked order` and `stock` of the net `Order Department` and `Delivery department` and places `receipted delivery note` and `delivered order` of the net `Delivery Department` and `Account department`. The resulting net `Logistics` is depicted in Fig. 4 (without net inscriptions).

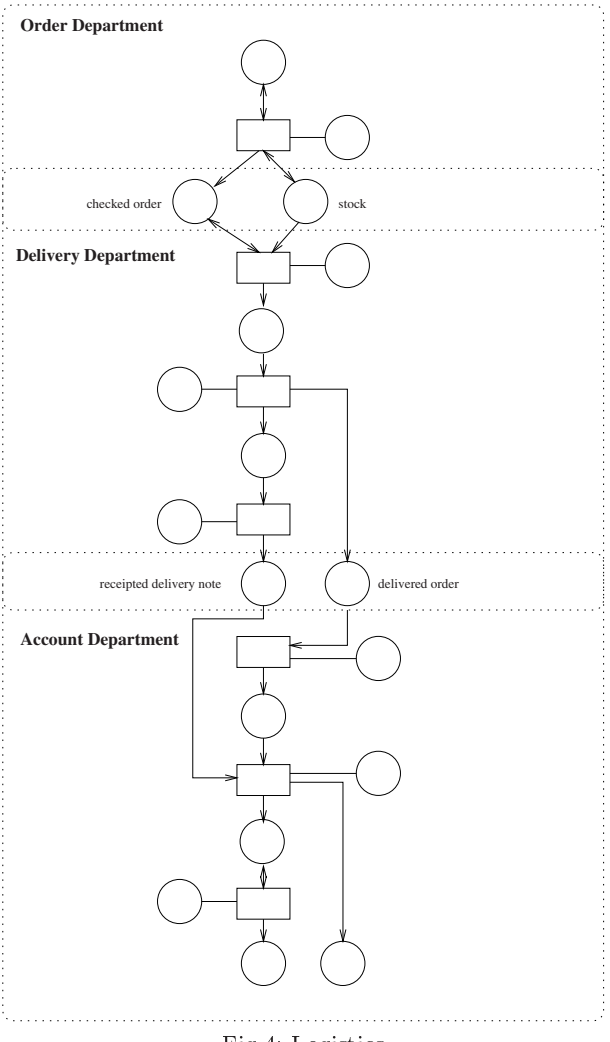

Fig.4: Logistics

Let us point out that in this section we only make limited use of the advantage of Algebraic Higher-Order nets compared with Algebraic High-Level Nets, because we use only one or two token on the higher-order places. But the notation as Algebraic Higher-Order nets will allow in Section 4 the flexible modeling

of business processes by adding additional token on higher-order places in Fig. 8 or changing the type of higher-order places in Fig. 9, without changing the net structure of the nets.

In the following we describe the three different Algebraic Higher-Order Nets `Order Department`, `Delivery department` and `Account department` depicted in Fig. 4 in more detail.

In Figure 5 the department of dealing with an order of a customer is depicted. Here the availability of ordered articles is checked by comparing the list of articles, which are available at the moment, with the list of ordered articles. We split the order into two parts, the current order including all articles, which are available at the moment, and a new order including all remaining articles to be carried out later as soon as they are available.

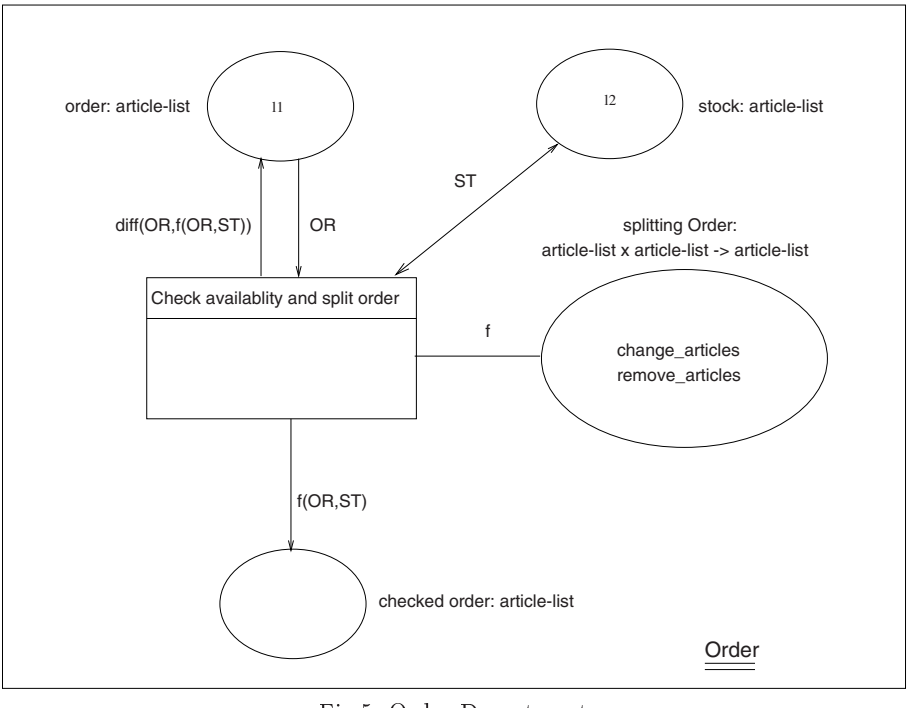

Fig.5: Order Department

The net `Order department` consists of three places called `stock`, `order` and `checked order`, where the type is given by the basic sort *article-list*. The initial marking l_1 of place `order` means that there is a list of articles ordered by a customer. Similar the marking l_2 of place `stock` represents a list of articles, which are available at the moment. Furthermore there is a higher-order place `splitting order` with function type *article-list* $\times$ *article-list* $\rightarrow$ *article-list* and a marking consisting of functions *change_articles* and *remove_articles* to split

an order. These higher-order functions *change_articles* and *remove_articles* and also the function *diff* are declared in the higher-order signature Order not shown explicitly. The function *remove_articles* can be interpreted by removing articles, which are not available at the moment, from the current order. These articles are postponed to the new order. In contrast the function *change_articles* means that the number of pieces of an ordered article is splitted into two parts. The first is the number of pieces available in the stock, i.e. l_2. The second is the difference of those in l_1 and l_2. So the article remain a part of the current order with quantities available in l_2 and becomes also a part of the new order with remaining quantities.

During the firing of transition `Check availability and split order` one of the functions *change_order* and *remove_order*, respectively, is selected by the assignment of the variable f. Furthermore the variable OR is assigned to the order l_1 and the variable ST is assigned to the stock l_2. Then we compute the current order $f(l_1, l_2)$, which is added to place `checked order`, while the new order *diff* $(l_1, f(l_1, l_2))$ is added to place `order` and consists of all remaining articles given by the difference of the original order l_1 and the current order $f(l_1, l_2)$.

In Figure 6 the Algebraic Higher-Order Net of the delivery department is depicted. In a first step articles of an order are removed from the stock and provided for loading up to trucks. Furthermore a delivery note is generated to inform the customer about articles to be received.

In the Algebraic Higher-Order Net `Delivery department` we have a closer look at the corresponding higher-order signature Delivery in this case, which is depicted below. It is used first of all to specify on one hand the construction of a list of articles and delivery notes. Due to the operation *make_article* articles consist of a pair of numbers of pieces *volume* and a specific article number *article-nr*. The list of articles is generated by the operation *no_list* for the empty list and the operation *add_article* to add recursively articles to an existing list. On the other hand the higher-order signature Delivery declares activities *remove_article*, *make_delivery_note* and *receipted_delivery_note*, which are carried out during the process in the order department. The meaning of these function is given in a corresponding higher-order algebra A of signature Delivery, which is not given explicitly in this paper.

$$
\begin{array}{ll}
Delivery = & Bool+ \\
basic\ \ sorts : & article, volume, article-nr, article-list, delivery-notes \\
opns : & make\_article : volume \times article-nr \to article \\
 & no\_list :\to article-list \\
 & add\_article : article-list \times article \to article-list \\
 & remove\_article : article-list \times article-list \to article-list \\
 & make\_delivery\_note : article-list \times bool \to delivery-note \\
 & receipted\_delivery\_note : delivery-note \to delivery-note \\
 & \ldots
\end{array}
$$

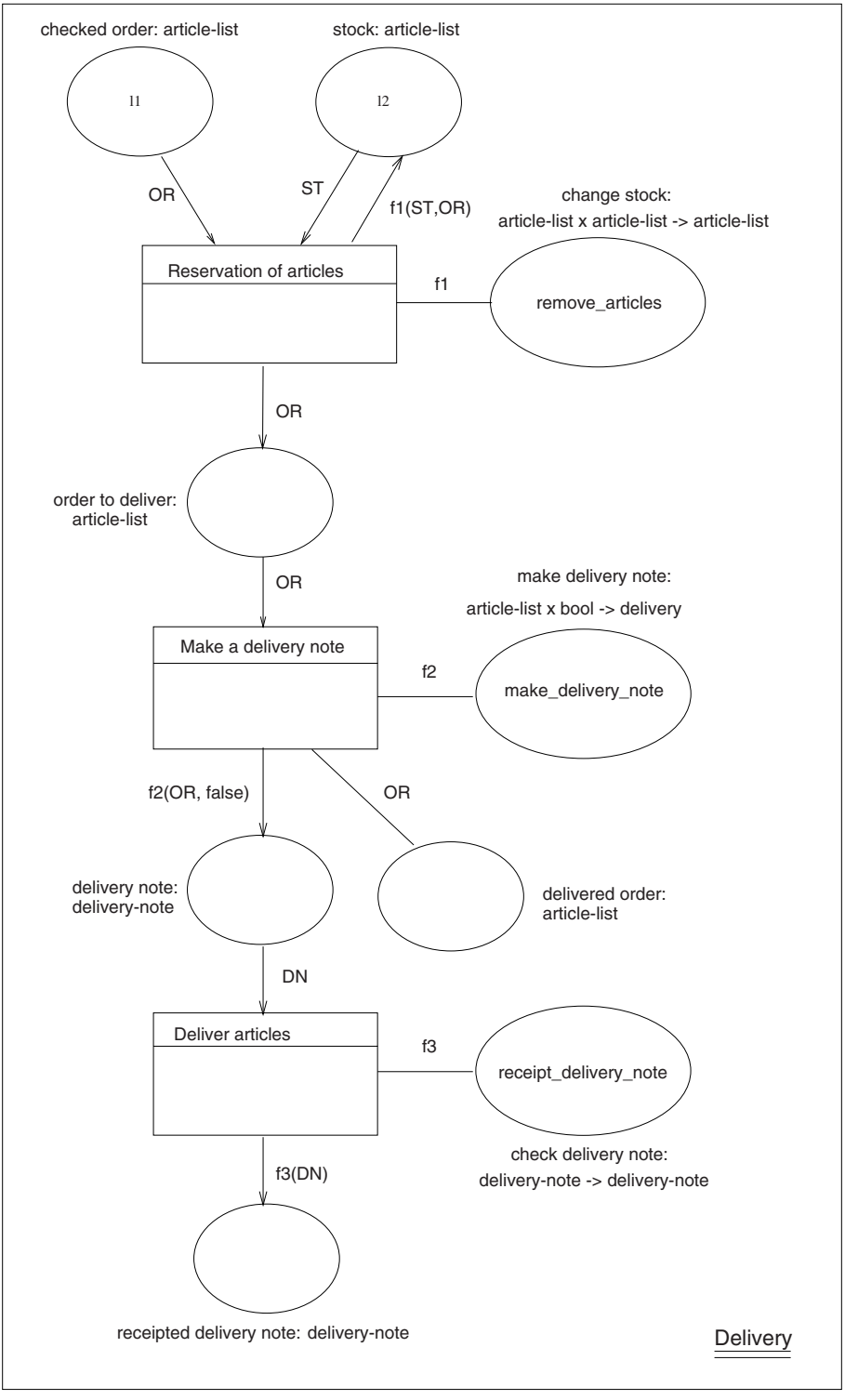

Fig.6: Delivery Department

The Algebraic Higher-Order Net `Delivery department` (see Fig. 6) consists of four places `stock`, `checked order`, `order to deliver` and `delivered order`, where the type is given by the basic sort *article-list*. Similar to the net `Order department`(see Fig. 5) the marking of the place `checked order` consists of a list of articles l_1 to define an order of a customer. Note, that the availability of ordered articles is already checked in the net `Order department` (see Fig. 5). Analogously the marking of the place `stock` consists of a list of articles l_2, which are available at the moment.

The type of places `delivery note` and `receipted delivery note` is given by the basic sort *delivery_note*, that is a lists of articles together with a boolean value. The boolean value *true* indicates that the delivery note is checked by customers. The type of the higher-order places `change stock`, `make delivery note` and `check delivery note` are given by corresponding higher-order function types with markings consisting of different activities carried out in the delivery department.

In the following we explain the firing behaviour of the Algebraic Higher-Order Net `Delivery department` in more detail. In order to fire transition `Reservation of articles` in Fig. 6 we have to give not only an assignment to variables ST and OR of type *article-list*, but also of variable f_1 of type *article-list* $\times$ *article-list* $\rightarrow$ *article-list*. Assigning l_1 to OR, l_2 to ST and f_1 to *remove-articles* we can fire transition `Reservation of articles`. The function *remove_articles* means, that ordered articles are removed from the current stock by changing the number of pieces. Then tokens l_1 and l_2 are consumed from place `checked order` and `stock`, respectively. On one hand token l_1 is added to the place `order to deliver`. On the other hand token $remove\_articles(l_2, l_1)$ is added to place `stock` and can be considered as the new stock, that is the original list of articles l_2 without ordered articles l_1. The marking of the higher-order place `change stock` remains to be unchanged as indicated by the horizontal line between the transition `Reservation of articles` and the higher-order place `change stock`.

In a next step a delivery note is generated by the application of the function *make_delivery_note* given as marking of place `make delivery note` to an order. In detail, we assign the order $l_3 = l_1$ to the variable OR and the function *make_delivery_note* to the variable f_2. During the firing of transition `Make a delivery note` we remove token l_3 from place `order to deliver` and add a token $l_4 = make\_delivery\_note(l_3, false)$ to place `delivery note` and a token l_3 to place `delivered order`. The token $make\_delivery\_note(l_3, false)$ can be interpreted as a list of ordered articles, which have to be checked by the customer.

Finally, during the firing of transition `Deliver articles` we assign the delivery note l_4 to the variable DN and the function *receipt_delivery_note* to the variable f_3. Then the delivery note l_4 is removed from place `delivery note` and is checked by the customer by changing the boolean value. This means that a token $receipt\_delivery\_note(l_4)$ is added to place `receipted delivery note`, while the marking of the higher-order place `check delivery note` left to be unchanged.

In Figure 7 we model the process of the account department. After articles are delivered an invoice is generated. In a next step the invoice is compared with the receipted delivery note. Once a week outstanding payments are checked and the account department reminds related customers.

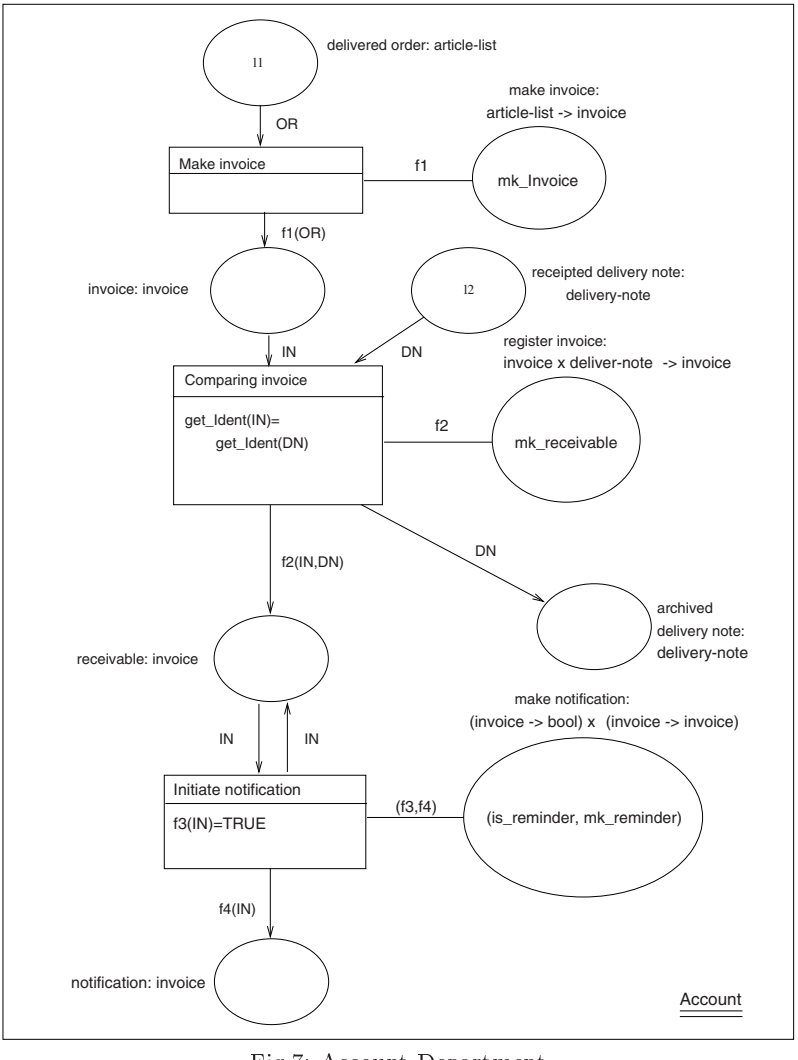

Fig.7: Account Department

4 Flexible Modeling of Business Processes

In this section we demonstrate the flexible modeling of business processes by introducing on one hand alternative activities and on the other hand a role model into the Algebraic Higher-Order Nets given in Section 3.

In Figure 7 we have modeled the usual process of the account department dealing with invoices. But in practice there are a lot of exceptions in the business process logistics, e.g. the delivery can be incomplete or not perfect with respect to their conditions. Here the customer can act in the following two ways: the order is canceled or he demands a credit note. The exceptions are introduced into the business process of the account department by an extension of the corresponding higher-order signature Account with activities concerning the cancelation and the credit note. We use the same net structure as given in Figure 7, but add tokens *mk_cancelation* and *mk_change* to place `register invoice`. Furthermore tokens (*is_cancelation*, *cancelation*) and (*is_credit*, *credit_note*) are added to place `make notification` and indicate, that in the account department cancelations and credit notes are generated in addition to the usual reminder (see Fig. 8).

The role model encompasses informations about persons, roles and corresponding activities. In [DG94] the role model is specified by a set of tables, showing roles allocated to activities and to persons, respectively. Table 1 and Table 2 give the role model for the business process `Order department` (see Figure 5). Here two different activities of splitting an order can be carried out by different persons: Maria, Marta and Martin can change the number of pieces of articles, while Maria and Markus can remove articles of an order.

Role	**Activity**
role 1	change articles and remove articles
role 2	change articles
role 3	remove articles

Table 1: Roles allocated to activities

Role	**Person**
role 1	maria
role 2	marta
role 2	martin
role 3	markus

Table2: Roles allocated to persons

To introduce the role model into our case study logistics, we extend the higher-order signature Order by a new basic sort *person* corresponding to person working in the company. The marking of the place `splitting order` is changed

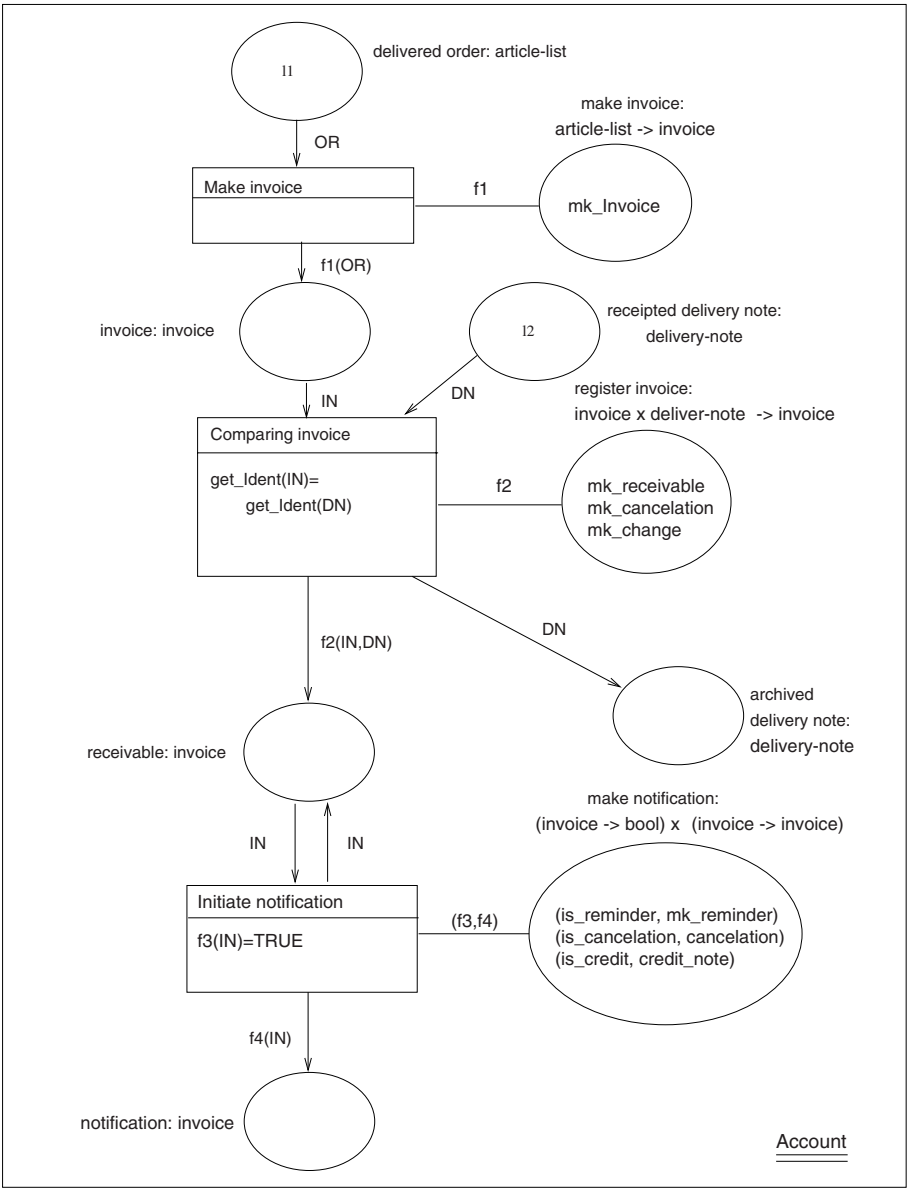

Fig. 8: Account Department with Alternative Activities

into a marking consisting of the product type *(article-list* $\times$ *article-list* $\rightarrow$ *article-list)* $\times$ *person* (see Figure 9) in contrast to the function type *article-list* $\times$ *article-list* $\rightarrow$ *article-list* in Figure 7.

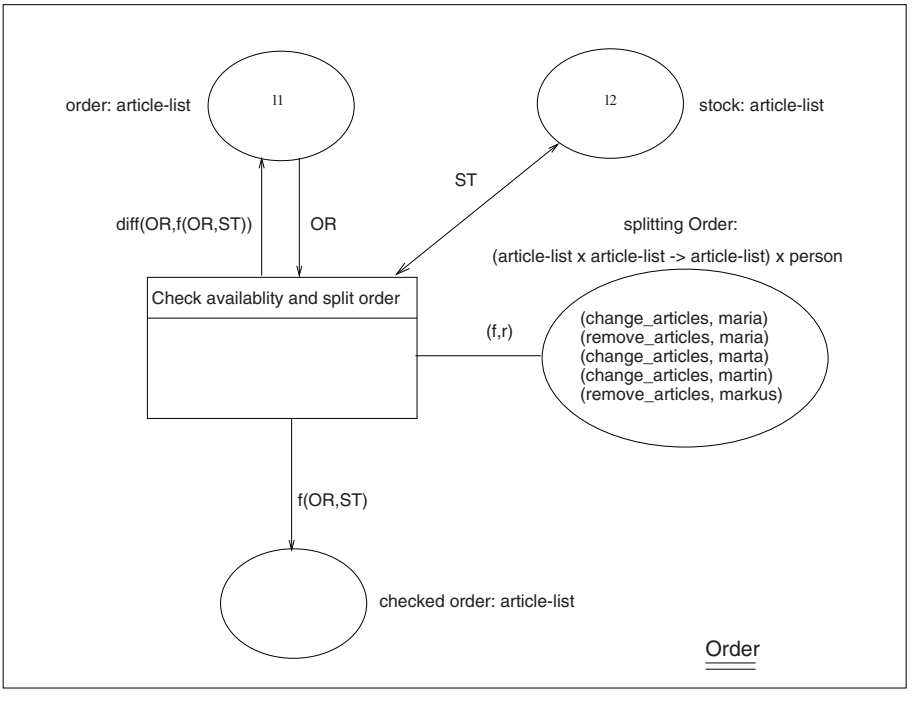

Fig.9: Order Department with Role Model

Another promising topic is the introduction of ownerships for documents to specify activity allocation constraints in the role model. These constraints can, for example, express that a person who has executed one activity to a certain document must also execute another specific activity. Similar to the procedure described above we change the basic sort of places with marking consisting of documents into a product type of persons and documents. For example we replace the basic sort of the place *order* by the product type *article-list* $\times$*person*. So the marking of the place *order* consists not only of orders given by a list of articles, but also of persons which have the ownership for modification of an order. Then in the equations of transition `Check availability and split order` we make sure that a list of articles can only be modified by a that person, which is addressed to the order.

5 Conclusion

We have presented Algebraic Higher-Order Nets leading to a more flexible and powerful modeling technique for processes. For software engineering purposes, especially in the area of business process engineering, this is an interesting approach, because Algebraic Higher-Order Nets support a flexible development of business process models without changing the net structure. We have demonstrated the use of Algebraic Higher-Oder Nets in our case study logistics on one hand by the abstraction of different activities (see Section 3) and on the other hand by the extension to exceptions and roles (see Section 4). The abstract view of the overall system modeling the cooperation of users, documents and data bases and the modeling of further departments are presented in our technical report [HS02].

Another promising topic is the extension not only by function types and product types, but also by predicates, partiality and sub sorting (according to higher-order specifications in [MHKB99]). But this remains for further research.

References

[DG94] W. Deiters and V. Gruhn. The FunSoft Net Approach to Software Process Management. *International Journal on Software Engineering and Knowledge Engineering*, 4(2):229–256, June 1994.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1985.

[EPR94] H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic High-Level Nets: Petri Nets Revisited. In *Recent Trends in Data Type Specification*, pages 188–206. Springer Verlag, 1994. Lecture Notes in Computer Science 785.

[Gen91] H.J. Genrich. Predicate/Transition Nets. In *High-Level Petri Nets: Theory and Application*, pages 3–43. Springer Verlag, 1991.

[Han97] Yanbo Han. *Software Infrastructure for Configurable Workflow System - A Model-Driven Approach Based on Higher-Order Nets and CORBA*. PhD thesis, Technical University of Berlin, 1997.

[Hof00] K. Hoffmann. Runtime Modifikation between Algebraic High Level Nets and Algebraic Higher Order Nets using Folding and Unfolding Construction. In G. Hommel, editor, *Communication-Based Systems, Proceedings of the 3rd International Workshop*, pages 55–72. TU Berlin, Kluwer Academic Publishers, 2000.

[HS02] K. Hoffmann and T. Schreiter. Case study logistics using algebraic higher-order nets. Technical report, Technical University Berlin, 2002.

[Jen92] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1: Basic Concepts. Springer Verlag, EATCS Monographs in Theoretical Computer Science edition, 1992.

[MHKB99] T. Mossakowski, A. Haxthausen, and B. Krieg-Brückner. Subsorted Partial Higher-Order Logic as an Extension of CASL. In *Recent Trends in Algebraic Development Techniques-14th International Workshop WADT'99*, pages 126–145, 1999.

[MR95] U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32, 1995.

[PER95] J. Padberg, H. Ehrig, and L. Ribeiro. Algebraic High-Level Net Transformation Systems. *Mathematical Structures in Computer Science*, 5:217–256, 1995.

[Sch94] A.-W. Scheer. *Business Process Engineering, Reference Models for Industrial Enterprises*. Springer-Verlag, Berlin, 1994.

[Val98] Rüdiger Valk. Petri Nets as Token Objects: An Introduktion to Elementary Object Nets. *Proc. of the International Conference on Application and Theory of Petri Nets*, 1998.