Toward a Comprehensive Approach to the Transformation of Logistic Models

Hans-Jörg Kreowski, Marco Franke, Karl Hribernik, Sabine Kuske, Klaus-Dieter Thoben and Caro von Totth

Abstract In this paper, we propose a framework for modeling of logistic systems with an emphasis on model transformation. Due to the complexity of logistic systems, their models are bound to consist of many heterogeneous components on various descriptive levels from the requirement definition to the platform-specific implementation. To cover these phenomena in a comprehensive way, our approach provides two main concepts: First, we introduce logistic models that may be the combination of a variety of component models, which in turn may be of different types, i.e., they may be specified by means of different modeling methods. Second, we offer model transformations that allow to translate logistic models of one type into logistic models of another type whenever needed (for example, to bridge the gap between visual platform-independent models and textual platform-specific models or to facilitate the interaction of component models of different types).

Keywords Interoperability · Model transformation · Heterogeneous modeling

H.-J. Kreowski (✉) · S. Kuske · C. von Totth
Universität Bremen, Linzer Strasse 9a, 28359 Bremen, Germany
e-mail: kreo@informatik.uni-bremen.de

S. Kuske
e-mail: kuske@informatik.uni-bremen.de

C. von Totth
e-mail: caro@informatik.uni-bremen.de

M. Franke · K. Hribernik · K.-D. Thoben
BIBA—Bremer Institut für Produktion und Logistik, Hochschulring 20,
28359 Bremen, Germany
e-mail: fma@biba.uni-bremen.de

K. Hribernik
e-mail: hri@biba.uni-bremen.de

K.-D. Thoben
e-mail: tho@biba.uni-bremen.de

H. Kotzab et al. (eds.), *Dynamics in Logistics*,
Lecture Notes in Logistics, DOI 10.1007/978-3-319-23512-7_11

Introduction

Today's logistic systems are often characterized by a widespread network of various processes and further components, like data bases and programming platforms, a dynamic structuring where subcomponents may be added, removed, or adapted to new requirements, many involved players with different interests, and fast changing customer requests, markets, and technologies. These phenomena result in quite heterogeneous logistic systems consisting of processes and further components that run on different platforms and are developed by means of various planning and modeling methods. For detailed information see Barnhart and Laporte (2006), Laguna and Marklund (2004), Recker (2006).

The increasing complexity, heterogeneity, and dynamism of current and future logistics systems mean that modeling and the transformation of models is highly relevant to the field of logistics. For example, the rapid cycles of contractual relationships in today's supply networks mean that logistics stakeholders frequently need to change their processes. This increasingly entails the integration of processes previously outside the scope of logistics providers, such as manufacturing processes. Furthermore, the integration of these processes into close, multi-stakeholder collaborations throughout supply networks means that multiple process models, often of different types, need to be integrated, simulated, and verified prior to and during contracts.

The IT systems employed by logistics providers also need to be able to handle this increasing complexity, heterogeneity, and dynamism. Currently, logistics providers often meet these demands with quick, in-house development of proprietary systems with little or no support for standard interfaces or data exchange formats. Current advances in cloud computing allow logistics stakeholders to completely outsource IT resources and use individual software modules as "cloud services" on demand on a pay-per-use basis. The resulting IT landscape is highly complex and distributed and spans multiple stakeholders across supply and retail chains. While standard interfaces and data exchange formats exist in the sector, their uptake by industry is not widespread. Correct, verifiable, and robust transformations between different data formats and interfaces, which themselves can be seen as models, are thus critical for the operation of today's logistics systems.

The demand for individualized products and services leads to an atomization of manufacturing and logistic operations. The result is the increasing need for logistics stakeholders to deal with high volumes of "batch size one" orders. Demands with regard to the quality, speed, and traceability of logistic operations are also rising. Recent technical and organizational developments in logistics have been made to meet these demands. The integration of auto-ID technologies such as RFID into logistic processes strives to manage individual items throughout supply and retail chains. With the Internet of Things, traceability can be extended to include information about the environment and condition of goods using sensors and embedded systems. Research into the autonomous control of individual logistics entities has been explored, for example, in the Collaborative Research Centre 637. "Autonomous

Cooperating Logistic Processes—A Paradigm Shift and its Limitations" (Hülsmann and Windt 2007; Hülsmann et al. 2011). Here, logistic objects with the capability to take decisions on their own are assumed to interact with each other in non-deterministic systems. The aim is to achieve increased robustness and positive emergence of the overall system due to the distributed and flexible handling of dynamics and complexity. Conventional methods of modeling are limited in their applicability to these types of highly complex and dynamic logistic systems. While a modeling methodology for autonomous control in logistics has been developed (Scholz-Reiter et al. 2011b), numerous challenges remain unaddressed (Scholz-Reiter et al. 2011a).

Cyber-physical systems (CPS) are the current culmination of these developments. A corresponding definition and detailed information are given in NSF National Science Foundation (2008) or Broy (2010). Their application is expected to revolutionize manufacturing and logistic processes—hence the title "Industrie 4.0" of the relevant high-tech strategy announced by the German government, which anticipates a fourth Industrial Revolution. CPS are themselves highly complex and dynamic "systems of systems" consisting of numerous computational and mechatronic devices. CPS components and their interactions are, however, currently represented by many different models spanning multiple domains so that suitable transformation approaches are required to achieve adequate views upon the systems and their components in design, engineering, and operation.

These trends lead to an increasing number of autonomous and heterogeneous systems in the application field of logistics. This evolution will increase the challenges in the interoperability of logistic processes regarding both the modeling and implementation of the underlying IT landscape. To enable the interoperability of logistic processes in the future, this paper presents an approach to how information can be exchanged between different modeling methods in design phase and between different IT systems in operation time. For this purpose, the notion of heterogeneous logistic model is given in section "Heterogeneous Logistic Models". Subsequently, a general and formal specification of transformation processes between different types of logistic models is given and illustrated with an example in section "Model Transformation Units". The example translates a specific type of business process model into Petri nets, in particular. Finally, the impact of such kind of representation forms and given transformation possibilities is described in the conclusion. The proposed modeling framework adapts earlier work in Kreowski et al. (2010, 2012), Kreowski and Kuske (2013) to the needs of logistics.

Heterogeneous Logistic Models

Models of logistic systems—in particular, large, distributed systems that support the cooperation of many parties—consist of many components that may be designed heterogeneously by means of different modeling methods. The components themselves may be structured in the same way. Without loss of generality,

the components may be ordered such that n components can be numbered from 1 to n and a model becomes a tuple of components. If a component is not structured itself, it can be specified as an entity or process of a modeling method or modeling language like *BPMN*, *UML* or Petri nets, or it is an elementary data object like a number, a symbol, a finite set, a string, a file or a document. Summarizing, we propose the following notion of (heterogeneous) logistic models.

Definition 1 Let $\mathcal{L}$ be a collection of modeling methods and modeling languages and let D be a set of data domains. Then a **logistic model** *mod* **of type** T is

1. $mod \in L$ for some $L \in \mathcal{L}$ with $type(mod) = L$,
2. $mod \in D$ for some $D \in \mathcal{D}$ with $type(mod) = D$,
3. a tuple $(mod_1, \ldots, mod_k)$ for some $k \in \mathbb{N}$ and a model mod_i of type T_i for $i = 1, \ldots, k$ with $type(mod) = T_1 \times \cdots \times T_k$.

The set of all models of type T is denoted by $\mathcal{M}(T)$.

To avoid distinction between these cases, in the following we consider all models as tuples. This is possible because there is no need to distinguish between a model *mod* and the 1-tuple (*mod*).

The underlying modeling framework is generic in that the modeling languages and the domains can be chosen according to the intended application and the taste of the designers. The following example may illustrate the principle.

Example 1 $\mathcal{L}$ may contain the modeling language *Business Process Model and Notation BPMN*, the *Unified Modeling Language UML*, the modeling methods of Petri nets, and of event-driven process chains. D may contain the integers, a set *ID* of identifiers, and the truth values $BOOL = \{true, false\}$. Then a sample model is the simple production process *producer* depicted in Fig. 1. It is of type $BPMN_{light}$ which is *BPMN* without pools and swim lanes (OMG 2013).

It can produce two products *A* and *B*, each of which can be sent whenever there is a respective order. The process *producer* is part of a supply chain with a trading

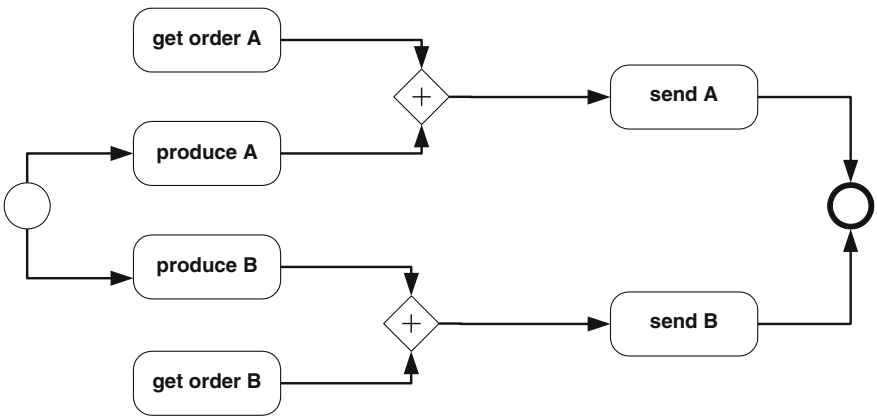

Fig. 1 The *producer* process in $BPMN_{light}$

process *trader* that puts orders of *A* and *B* to the *producer* and receives the sent products from there. In turn, it gets orders from a consumer process *consumer* that also receives the products sent by the *trader*. The start event triggers the activities *get order A* and *get order B* only if there are such orders in the environment, i.e., put by the *consumer*. Finally, *consumer* can put orders to *trader* and receive the products from there. The processes *trader* and *consumer* may also be modeled in $BPMN_{light}$ as given in Figs. 2 and 3 respectively.

If we consider the two sets of activities *InOut* = {*get order A,get order B,send A, send B*} and *OutIn* = {*put order A,put order B,receive A,receive B*}, then the communication between *producer* and *trader* on one hand and between *trader* and *consumer* on the other hand can be expressed by the pairs (*put order A,get order A*), (*put order B,get order B*), (*send A,receive A*), (*send B,receive B*). The set *connect* of these four pairs is a model of the set type with elements of the type *InOut* × *OutIn*. The combination

$$supply_0 = (producer,\ connect,\ trader,\ connect,\ consumer)$$

models the whole supply chain as a quintuple of type $BPMN_{connect} = BPMN_{light} \times C \times BPMN_{light} \times C \times BPMN_{light}$ where *C* is the type of *connect*.

To complete the section of logistic models, one further aspect is important. As long as one considers free tupling, the components are unrelated with each other so that the tuple and its separate components provide the same information. But in many practical cases the components are related—and should be related—in some way. For example, the $supply_0$-tuple only makes sense if the first set of connectors connects activities of *producer* and *trader* while the second set connects activities of *trader* and *consumer*. Therefore, we allow adding conditions to the declaration of model types restricting the class of models. To formulate the conditions, called

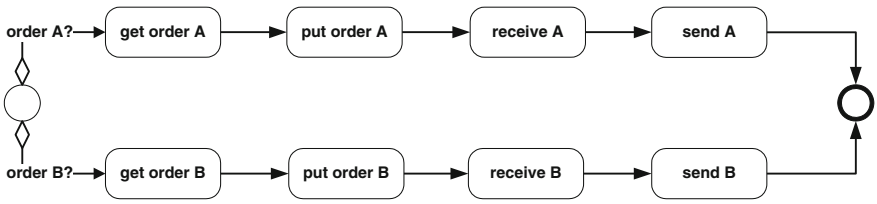

Fig. 2 The *trader* process in $BPMN_{light}$

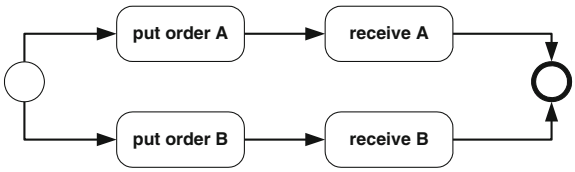

Fig. 3 The consumer process in $BPMN_{light}$

constraints in the following, one may assume a proper logic like the propositional calculus or first order logic. More practically speaking, constraints may be written like Boolean expressions in programming languages.

Definition 2 Let $T = T_1 \times \cdots \times T_k$ be a type with types T_i for $i = 1, \ldots, k$. Let x be some syntactic entity that describes a property that may hold for models of type T or not. Then x is called a **constraint** and T *with* x a **constraint type**. The set of all models of type T for which x holds is denoted by $\mathcal{M}(T \textit{ with } x)$.

It should be noted that constraints can be combined by Boolean operators like *and* and *or* with the obvious meaning that *and* yields the intersection and *or* the union respectively. If we assume the constraint *true* that always holds, then the type T and the constraint type T *with true* specify the same set of models. Hence, there is no need to distinguish between types and constraint types. In the following, the term type includes constraint types.

Model Transformation Units

As discussed in the Introduction, there are various good reasons, if not necessities, to transform logistic models. First of all, visual models must be transformed into programs to be integrated into a running logistic system. Moreover, one may want to check required properties using some model checker. But the input models of the respective tool may be of a different type than the models at hand. Model transformation can solve the problem. In this section, we introduce the notion of model transformation units that allow transforming logistic models as introduced in the previous section. As a logistic model is a tuple of component models, the components can be transformed componentwise and simultaneously by means of actions. An action specifies for each component how it is processed using operations that are available for the models of the respective types. If the component models are numbers, strings, or sets, then one can use arithmetic, word-processing, or set-theoretic operations respectively. If the component models are modeled according to a logistic modeling language or method, then suitable operations must be chosen for the construction, reconstruction, and deconstruction of the models. If, finally, the component models are tuples again, then the component operation can be recursively chosen as an action.

As actions keep the type of models, their application can be iterated. In this way, a set of actions defines a complex model transformation. But, usually, transformation processes are not just arbitrary sequences of action applications starting and ending on arbitrary models. Therefore, we assume in addition that initial and terminal models can be specified and that the order of action application can be restricted by means of a control condition.

There is one further aspect to be considered. While actions preserve the type of the processed model, model transformations are meant to transform input models

into output models which have different types usually. For example, a visual model of $BPMN_{light}$ may be transformed into a Petri net or a *JAVA* program so that not only the models change, but also their types. To cover this aspect, input, output, and working types can be chosen separately due to the intended model transformation. Then the input models are adapted by an initialization to models of the working type on which the actions run. Finally, the resulting working models are projected to the output type by a terminalization. Given an input model, some components of the working type can be components of the input models, while others may be auxiliary or needed as output components. They are chosen as fixed constant initial models. Given a resulting working model, some of its components are taken as the output model. This leads to the following definition:

Definition 3 Let $\mathcal{L}$ be a collection of modeling methods and modeling languages and $\mathcal{D}$ be a set of data domains. Let, for each $X \in \mathcal{L} \cup \mathcal{D}$, OP_X be a set of unary operations on the models of type X. Then a **model transformation unit** is a system $mtu = (ITD, OTD, WT, A, C)$ where

- $WT = T_1 \times \cdots \times T_k$ is the **working type**,
- A is a set of **actions** on WT,
- C is a **control condition**,
- ITD is the **input type declaration** consisting of an **input type** $IT = I_1 \times \ldots. \times I_m$ *with* x and an **initialization** *initial*,
- OTD is the **output type declaration** consisting of an **output type** $OT = O_1 \times \ldots. \times O_n$ *with* y and a **terminalization** *terminal*.

subject to the following conditions:

1. each action has the form $a = (op_1, \ldots.., op_k)$ *with* z where $op_i \in OP_{T_i}$ for $i = 1, \ldots.., k$ *and* z is a constraint,
2. *initial* associates each working type component T_i with some input type component I_j or with a fixed model of type T_i,
3. *terminal* associates each output type component O_j with some working type component T_i.

To enhance the flexibility of actions, we assume that the set of operations OP_T for each type T contains the void operation "-", which refers to the identity. Consequently, an action keeps a component of a model invariant if the respective component of the action is void.

The following example may illustrate the features of model transformation units.

Example 2 We would like to transform a $BPMN_{light}$ model like *producer* into a Petri net, the type of which is denoted by *PN*, to enable us—for example—to employ a model checker for Petri nets (see Aalst and Stahl 2011; Hee et al. 2013 for further relations between business process models and Petri nets). Therefore, the input type is $BPMN_{light}$ and the output type is *PN*. As working type, we take the product $BPMN_{light} \times PN$. The initialization assigns $BPMN_{light}$ to itself and *PN* to the empty Petri net $\varnothing$. Therefore, the initial working models are pairs of $BPMN_{light}$ models and $\varnothing$ like (*producer*, $\varnothing$). The terminalization assigns the only output type

PN to the *PN*-component of the working type. Consequently, the Petri net of any resulting working model is considered as output model. To specify the dynamic part, we need operations and actions. The basic idea is to replace each flow object f of the initial $BPMN_{light}$ model by a place pf, a transition tf and a flow relation from tf to pf as well as each sequence flow from a flow object f to a flow object f' by a flow relation between pf and tf'. In the case of the end event, the transition must be doubled with a flow to the end place each and the two sequence flows into the end event must be redirected to the now different end transitions. To achieve this, we need an operation *mark* on $BPMN_{light}$ models that mark flow objects and sequence flows as *done* provided that they are not yet marked and operations *add*(f) and *add*($f \rightarrow f'$) on Petri nets where *add*(f) adds $tf \rightarrow pf$ to a given Petri net and *add*($f \rightarrow f'$) adds an edge from the place pf to the transition tf' provided that both exist. This allows us to combine these operations to the actions *act*(f) = (*mark*(f), *add*(f)) and *act*($f \rightarrow f'$) = (*mark*($f \rightarrow f'$), *add*($f \rightarrow f'$)) for some identifiers $f, f' \in ID$. It should be noted that the actions can only be applied if the parameters are flow objects and sequence flows of the input process and that none of them can be applied twice so that the length of every sequence of action applications is bounded by the number of flow objects and sequence flows. Moreover, no action can be applied if all elements of the input process are marked by *done*. If we require as control condition that *act*(f) and *act*(f') be applied before *act*($f \rightarrow f'$) and that actions be applied as long as possible, then each sequence flow becomes reflected in the flow relation of the corresponding Petri net and all elements of the input process are carried over to the Petri net part. For example, the input model *producer* is transformed into the following Petri net, which is shown in Fig. 4.

A schematic representation of the sample model transformation unit may look as follows.

BPMN*$_{\text{light}}$ -to-*PN
input: $BPMN_{\text{light}}$
working: $BPMN_{\text{light}} \times PN$
initial: $BPMN_{\text{light}} \nrightarrow BPMN_{\text{light}}$ & $PN \nrightarrow \emptyset$
actions: act(f) = (mark(f), add(f))
act(f ➔ f') = (mark(f ➔ f'), add(f ➔ f') for f, f'∈ ID
control: act(f) > act (f ➔ f') & act (f') > act(f ➔ f') & as long as possible
output: *PN*
terminal: $PN \nrightarrow PN$

As the sample model transformation unit transforms $BPMN_{light}$ processes into Petri nets, a model transformation unit relates input models to output models in general. A given input model induces an initial working model due to the input type declaration. The working model is transformed by a sequence of action applications which is regulated by the control condition. In particular, the control condition specifies when the action application can terminate. Then the reached working model induces an output model due to the output type declaration.

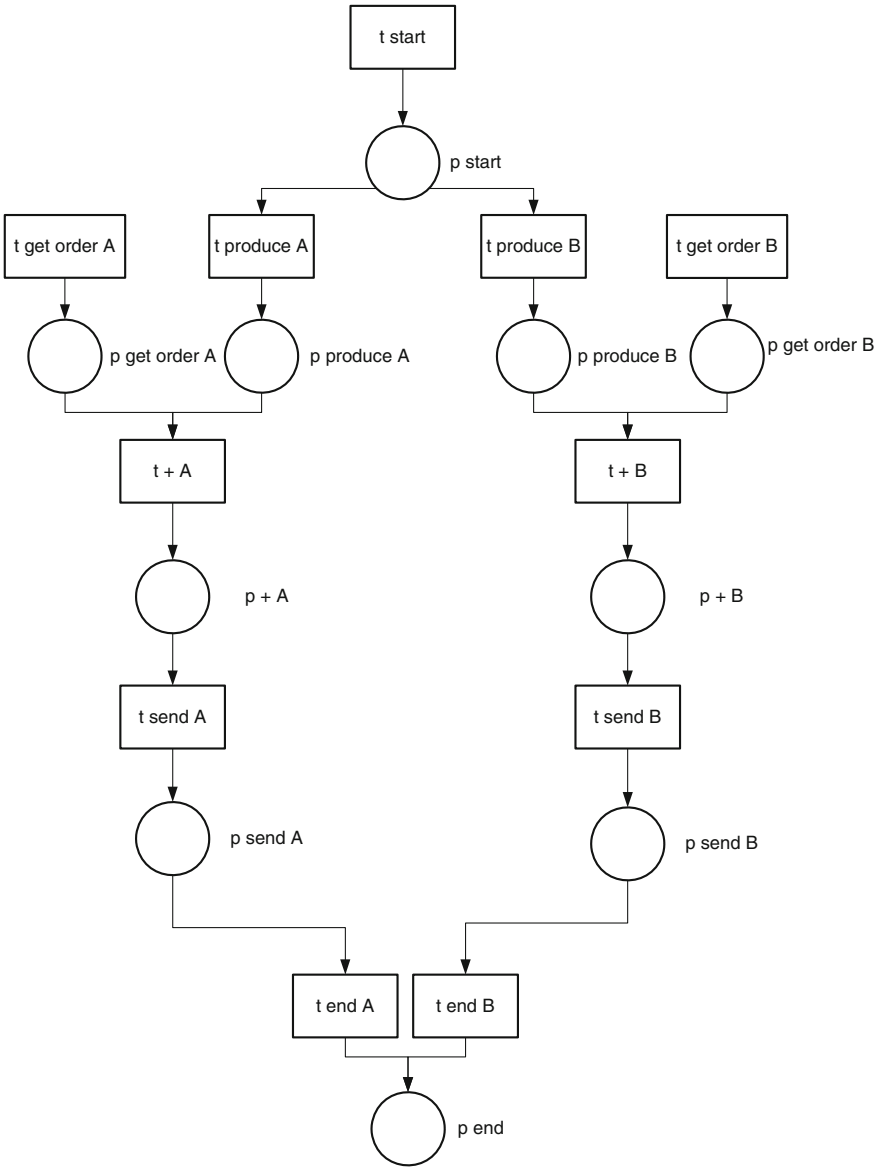

Fig. 4 The $BPMN_{light}$ process *producer* transformed into a Petri net

Definition 4 Let *mtu* = (*ITD, OTD, WT, A, C*) be a model transformation unit. Then *mtu* specifies the **semantic relation** $SEM(mtu) \subseteq \mathcal{M}(IT) \times \mathcal{M}(OT)$ between input and output models, where an input model $in \in \mathcal{M}(IT)$ is transformed into an output model $out \in \mathcal{M}(OT)$, i.e. $(in, out) \in SEM(mtu)$, in the following three steps:

1. Let $IT = I_1 \times \cdots \times I_m$ *with* x and $WT = T_1 \times \cdots \times T_k$. Then $in = (in_1, \ldots, in_m)$ gives rise to a working model $mod(in) = (mod_1, \ldots, mod_k)$ with $mod_i = in_j$ if *initial* associates T_i with I_j, and $mod_i = init_j$ if *initial* associates T_i with the fixed model $init_j$.
2. Let $\Rightarrow_A \subseteq \mathcal{M}(WT) \times \mathcal{L}(WT)$ denote the application of actions in A to working models, $\Rightarrow_A{}^* \subseteq \mathcal{L}(WT) \times \mathcal{M}(WT)$ the reflexive and transitive closure of $\Rightarrow_A$, i.e., the arbitrary iterations of action applications, and $\Rightarrow_{A,\,C}{}^* \subseteq \mathcal{M}(WT) \times \mathcal{M}(WT)$ the iterated action applications that obey the control condition C. Then $mod(in)$ is transformed using $\Rightarrow_{A,\,C}{}^*$. A working model $mod' \in \mathcal{M}(WT)$ is considered as a result if $mod \Rightarrow_{A,\,C}{}^* mod'$.
3. Let $mod' = (mod'_1, \ldots, mod'_k)$ be a result and $OT = O_1 \times \cdots \times O_n$ *with* y. Then the output $out = (out_1, \ldots, out_n)$ is given by $out_j = mod'_i$ if *terminal* associates O_j with T_i provided that *out* satisfies the constraint y.

A further example may help to see the meaning and significance of the introduced concepts.

Example 3 The supply chain $supply_0$ in Example 1 is a heterogeneous model with five components of two different types. It may be preferable to have a homogeneous model of some suitable type, say *BPMN*, because there may be a simulator available for *BPMN* processes or an automatic transformation of *BPMN* processes into *JAVA* programs. A model transformation unit can bridge the gap between $supply_0$ and *BPMN*.

BPMN*$_{\text{connect}}$ -to-*BPMN
input: $BPMN_{\text{connect}}$
working: $BPMN_{\text{connect}} \times BPMN$
initial: $BPMN_{\text{connect}} \rightarrowtail BPMN_{\text{connect}}$ and $BPMN \rightarrowtail \varnothing$
actions: *(-,add(p)) & (-,add(c))*
control: (-,*add*(*p1*)); (-,*add*(*p2*)); (-,*add*(*p3*)); (-,*add*(*c1*)); (-,*add*(*c2*))
output: *BPMN*
terminal: $BPMN \rightarrowtail BPMN$

where $\varnothing$ denotes an empty pool, the operation $add(p)$ for a $BPMN_{light}$ process p adds p as a new swim lane to the given pool, the operation $add(c)$ for a binary relation $c \subseteq ID \times ID$ adds the elements of c as message flows provided that c relates flow objects of two swim lanes of the given pool, and where $(p_1, c_1, p_2, c_2, p_3)$ identifies the input model. Moreover, the control condition requires that the five actions are applied one after the other denoted by means of the sequencing operator ";". Semantically, the model transformation unit takes a $BPMN_{connect}$ model, combines it with the empty pool in the first step, applies the five actions, and projects the result to the second component. Note that the first component is not changed and that the model constraints make sure that the two connectors c_1 and c_2 relate activities of the respective swim lanes. Applied to $supply_0$, the transformation yields the *BPMN* process $supply_1$, which is shown in Fig. 5, where dashed arrows

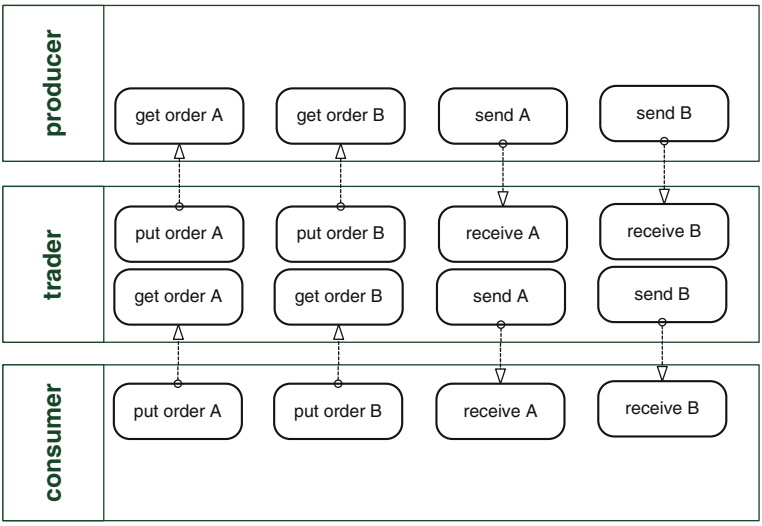

Fig. 5 *BPMN* model of the supply chain (confer example 1 for the complete subprocesses)

denote the message flows and those activities of the three swim lane processes that are involved in the communication are given explicitly.

Conclusion

In this paper, we have sketched fundamental concepts constituting a framework for the modeling of logistic systems: logistic models that can be heterogeneously composed of component models and specified by means of different modeling languages as well as model transformations that bridge the gap between different descriptive levels and support the interaction of models of different types. To shed more light on the significance and usefulness of the approach, further topics must be studied:

1. Further case studies, which are more realistic than our small toy supply chain, are needed. In particular, the use of further modeling languages and methods and their coexistence within modeling of one model should be demonstrated.
2. We have pointed out that model transformation is necessary if one wants to employ a tool for testing, simulation, visualization, or verification that requires input models of another type than the models at hand. It would be of interest to show explicit cases where such transformations are advantageous.
3. The actions of model transformation units combine operations on the component models depending on their types. If the types refer to truth values, numbers, sequences, or sets, then one can use the usual Boolean, arithmetic, word-processing, or set-theoretic operations respectively. If a component is a tuple again, then it can be operated by actions again. But if it is specified by

means of a modeling language, then it may be necessary to enrich the language by new operations like the marking of activities and sequence flows in the $BPMN_{light}$ examples and the building operations in the *BPMN* and Petri net examples.

4. The introduced modeling of logistic systems has not only a formal syntax, but also a precise formal semantics. Potentially, this permits to prove interesting properties like termination, functionality, and correctness of model transformations, which cannot be addressed properly without formal semantics.

References

Barnhart C, Laporte G (2006) Handbooks in operations research and management science: transportation. Elsevier Science and Technology, Amsterdam

Broy M (2010) Cyber-physical systems: innovation durch softwareintensive eingebettete Systeme. Springer, Berlin

Hee K, Sidorova N, Werf J (2013) Business process modeling using petri nets. In: Jensen K, van der Aalst WMP, Balbo G, Koutny M, Wolf K (eds) Transactions on petri nets and other models of concurrency lecture, Notes in Computer Science 7480:116–161

Hülsmann M, Windt K (2007) Understanding of autonomous cooperation and control in logistics —the impact of autonomy on management, information, communication and material flow. Springer, Berlin

Hülsmann M, Scholz-Reiter B, Windt K (2011) Autonomous cooperation and control in logistics. Springer, Berlin

Kreowski H-J, Kuske S (2013) Graph tuple transformation. Electronic Communications of the EASST 62, 23 pages

Kreowski H-J, Kuske S, von Totth C (2010) Stepping from graph transformation units to model transformation units. Electronic Communications of the EASST 30, 24 pages

Kreowski H-J, Kuske S, von Totth C (2012) Combining graph transformation and algebraic specification into model transformation. In: Mossakowski T, Kreowski H-J (eds) Proceedings of the international workshop on algebraic development techniques (WADT 2010) Lecture Notes in Computer Science, vol 7137. pp 193–208

Laguna M, Marklund J (2004) Business process modeling, simulation, and design. Pearson/Prentice Hall, New Jersey

NSF National Science Foundation (2008) Cyber-physical systems. Program announcements and information. NSF 08–11, http://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf08611. Accessed 16 Oct 2013

OMG Object Management Group (2013) Business process model and notation (BPMN), http://www.omg.org/spec/BPMV/2.02/pdf. Accessed 10 Mar 2013

Recker JC (2006) Process modeling in the 21st century. BP Trend, pp 1–6

Scholz-Reiter B, Rippel D, Sowade S (2011a) Limitations in modeling autonomous logistic processes—challenges and solutions in business process modeling. In: Proceedings of the IEEE international symposium on assembly and manufacturing 2011 (ISAM'11). Tampere, USB-Proceedings, 6 pages

Scholz-Reiter B, Sowade S, Rippel D (2011b) Modeling the control system infrastructure for autonomous logistics processes. In: Duffy NA (ed) 44th CIRP conference on manufacturing systems. Omnipress, Madison-Wisconsin, pp 1–6

Van der Aalst W, Stahl C (2011) Modeling business processes: a petri net-oriented approach. Cooperative information systems. MIT Press, Cambridge